

AD-A141 467

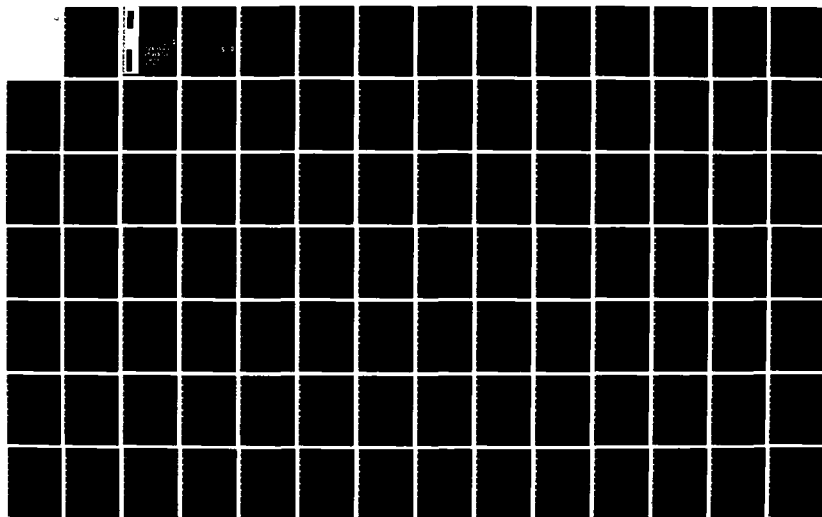
A STUDY OF THE APPLICATION OF SOFTWARE METRICS TO COBOL 1/2  
(U) OHIO STATE UNIV COLUMBUS COMPUTER AND INFORMATION  
SCIENCE RES. N C DEBNATH ET AL. JUL 83

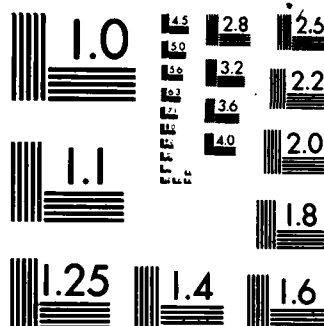
UNCLASSIFIED

OSU-CISRC-TR-84-3 ARO-17150.5-EL

F/G 9/2

NL





AD-A141 467

DTIC FILE COPY

TECHNICAL REPORT SERIES

OSU-CISRC-TR-84-3

A STUDY OF THE APPLICATION  
OF SOFTWARE METRICS TO COBOL

By

N. C. Debnath and S. H. Zweben

This document has been approved  
for public release and sale; its  
distribution is unlimited.

DTIC  
ELECTE  
MAY 22 1984  
S A D

COMPUTER &  
INFORMATION  
SCIENCE  
RESEARCH CENTER

84 05 21 189

THE OHIO STATE UNIVERSITY COLUMBUS, OHIO

A STUDY OF THE APPLICATION  
OF SOFTWARE METRICS TO COBOL

By

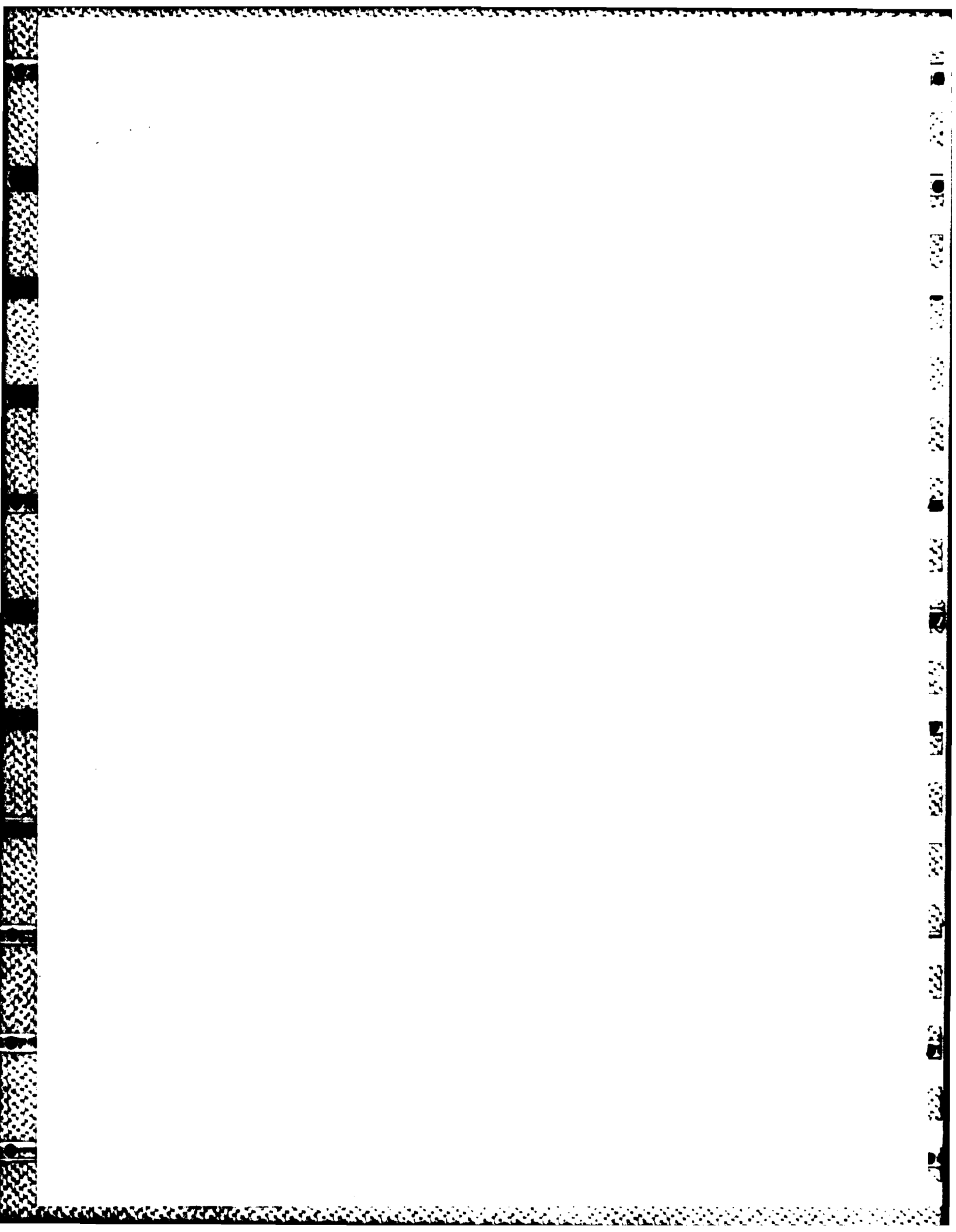
N. C. Debnath and S. H. Zweben

DTIC  
ELECTE  
S MAY 22 1984 D  
A

This document has been approved  
for public release and sale; its  
distribution is unlimited.

Computer and Information Science Research Center  
The Ohio State University  
Columbus, Ohio 43210

July 1983



Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ARO 17150.5-EL	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER <b>A141467</b>
4. TITLE (and Subtitle)  A Study of the Application of Software Metrics to COBOL		5. TYPE OF REPORT & PERIOD COVERED  Technical Report
7. AUTHOR(s)  N. C. Debnath, S. H. Zweben		6. PERFORMING ORG. REPORT NUMBER
9. PERFORMING ORGANIZATION NAME AND ADDRESS  Ohio State Univ		8. CONTRACT OR GRANT NUMBER(s)  DAAG29-80-K-0061
11. CONTROLLING OFFICE NAME AND ADDRESS U. S. Army Research Office Post Office Box 12211 Research Triangle Park, NC 27709		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS  N/A
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		12. REPORT DATE July 1983
		13. NUMBER OF PAGES 145
		15. SECURITY CLASS. (of this report)  Unclassified
16. DISTRIBUTION STATEMENT (of this Report)  Approved for public release; distribution unlimited.		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES  The view, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)  Computer Programming Computer Programs Software Engineering		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number)		

This report presents the results of a study of software metrics applied to various classes of COBOL programs. Particular attention is given to the software science metrics of Halstead, which were applied to hundreds of COBOL programs written by students at Ohio State University, as well as several

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

ARO 17150.5-EL

20. ABSTRACT CONTINUED:

production COBOL programs. The results include support of the inclusion of Data Division in the software science counting strategy, nonsupport for the use of the software science language level metric, and the identification of a weakness in the ability of the Halstead E measure to capture integration effort.

Several proposed complexity metrics were compared for their ability to predict actual development effort, with none of the metrics studied behaving in an impressive manner. Some approaches for refining existing complexity metrics to overcome their apparent weaknesses are suggested.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

## Table of Contents

<b>1. Introduction</b>	1
<b>2. Verification of the Software Science Metrics in COBOL Environment</b>	5
2.1 Review of the Software Science Metrics	5
2.2 Analysis of Students' Programs	7
2.3 Analysis of University System Computer Center Programs	49
2.4 Comparison of the Results Between the OSU Analyzer and the Purdue Analyzer	66
<b>3. Relationships Among Various Software Metrics</b>	71
3.1 Motivation	71
3.2 Background and Definitions of Metrics	71
3.3 Source of Data and the Comparison of the Metrics	78
3.4 Module Based Comparison Between Effort and Information Flow Complexity	80
<b>4. Development of a New Approach to Measuring Software Effort</b>	89
4.1 Motivation	89
4.2 Formulation of the Approach	91
4.2.1 Strategy 1	92
4.2.2 Strategy 2	94
4.3 Preliminary Results	96
4.4 Further Refinement	98
<b>5. Conclusion</b>	101
<b>References</b>	103
<b>I. A</b>	105
1.1 Counting Strategy Used to Find the Information Flow Complexity for COBOL Programs	105
1.2 Calculations for Finding Information Flow Complexity for Each Program	107
1.3 Calculations for Finding Chunk Model Complexity for Each Program	107
<b>II. B</b>	121




Accession For	
GRA&I	<input checked="" type="checkbox"/>
TAB	<input type="checkbox"/>
Announced	<input type="checkbox"/>
Notification	
Distribution/	
Availability Codes	
Avail and/or	
Special	
A-1	



## Abstract

This report presents the results of a study of software metrics applied to various classes of COBOL programs. Particular attention is given to the software science metrics of Halstead, which were applied to hundreds of COBOL programs written by students at Ohio State University, as well as several production COBOL programs. The results include support of the inclusion of Data Division in the software science counting strategy, nonsupport for the use of the software science language level metric, and the identification of a weakness in the ability of the Halstead E measure to capture integration effort.

Several proposed complexity metrics were compared for their ability to predict actual development effort, with none of the metrics studied behaving in an impressive manner. Some approaches for refining existing complexity metrics to overcome their apparent weaknesses are suggested.



## Preface

This report is the result of research supported in part by the U.S. Army Research Office of Scientific Research under contract DAAG29-80-k-0061. It is being published by the Computer and Information Science Research Center (CISRC) of the Ohio State University in conjunction with the Department of Computer and Information Science. CISRC is an interdisciplinary research organization which consists of the staff, graduate students, and faculty of many University departments and laboratories.

Index Terms

Software Engineering, Software Metrics, Development Effort

## 1. Introduction

Software Metrics, as a branch of Software Engineering, plays an important role in the analysis and evaluation of software. Many complexity metrics for computer programs have been developed. There are two categories into which many of the more popular metrics can be divided. The first category may be termed lexical metrics, which are based on the counts of various lexical tokens in the system. This category includes Halstead's software science metrics [3] and McCabe's cyclomatic complexity metrics [5]. The second category of metrics deals with the system connectivity by observing the flow of control or information among the system components. Recent work of Henry and Kafura [4] using information flow, and the chunk model complexity measure by Davis [1] fall into this category.

This report presents the results of a study of the software metrics with special emphasis on Halstead's software science metrics. The area of software science has been explicitly studied by many independent research groups. Since many of the experimental results reported by Halstead and others have been very encouraging, these metrics have received considerable attention from the computer science community. Most of the work in applying metrics of computer software using the methodology of software science has concentrated on relatively few programming languages such as Fortran and PL/I. COBOL has received relatively little research attention with two notable exceptions. Zweben and Fung [11] reported the results of a preliminary study of COBOL programs which were counted manually. The work of Zweben and Fung [11] initiated the writing of a software science analyzer [2] for further study in software science metrics in a COBOL environment. The use of this analyzer helps to collect a large amount of data on COBOL programs. The analyzer provides a mechanical way of counting the tokens (operators and operands) of a COBOL program, and hence can produce all of the software science statistics. The software metrics research group at Purdue University has done perhaps the

most comprehensive study of COBOL programs using software science [6]. Some aspects of our study, particularly those described in the next chapter, are similar to theirs, though different programs, programmers and analyses were performed. This report will present the results of the analysis of a very large number of COBOL programs collected from various sources.

This report is divided into five main chapters. The next chapter deals with the verification of the software science metrics using a large number of COBOL programs. The chapter has been divided into four sections. A brief review of software science [3] is presented in the first section. The second and third sections are concerned with the results of the analysis of the COBOL programs collected from two different sources, namely undergraduate students' programs at The Ohio State University (OSU) and COBOL programs written by production programmers at the University Systems Computer Center of OSU. The last section shows the result of the comparison of the software science statistics for some COBOL programs, which were run through two different COBOL analyzers -- one developed at OSU and the other produced by the software science research group at Purdue University. The third chapter has been included to show the relationships among four different software metrics, namely Halstead's Effort metric, McCabe's cyclomatic complexity metric, Kafura's information flow complexity metric and Davis' chunk model complexity metric. The primary motivation of this chapter is to study the concept of effort and its relation to development time of software. The four different complexity metrics were evaluated for a small set of COBOL programs in order to assess the relative complexities of these programs. The summary of the results of comparing these four metrics is shown at the end of this chapter. The weaknesses of these metrics motivated consideration of a new model for software effort. This is outlined in Chapter 4. Finally, the closing chapter suggests additional work to be done on this topic.

Two appendices have been included for completeness of the report. Appendix A describes the counting strategy used to calculate the information flow and chunk model complexities for COBOL programs, together with the explicit calculations required for the results of Chapter 3. Appendix B gives the major steps of the calculations carried out to obtain the results shown in Chapter 4.

## 2. Verification of the Software Science Metrics in COBOL Environment

This chapter deals with the verification of the software science metrics using a large number of COBOL programs collected from various sources.

### 2.1 Review of the Software Science Metrics

In software science, a computer program is considered to be a string of tokens which are divided into "Operators" and "Operands". Generally, any symbol or keyword group in a program that specifies an algorithm action of the computer is considered an operator, and any symbol used to represent data is considered an operand. All software science measures are functions of the counts of the operators and the operands.

The basic metrics in software science are defined as:

- $n_1$  = number of unique operators
- $n_2$  = number of unique operands
- $N_1$  = total occurrences of operators
- $N_2$  = total occurrences of operands.

The length of the program is defined as

$$N = N_1 + N_2 \quad (2.1)$$

and the vocabulary of a program is defined as:

$$n = n_1 + n_2. \quad (2.2)$$

All other metrics are defined in terms of these basic terms and are shown below.

The estimated length is defined by the length equation:

$$\hat{N} = n_1 \log n_1 + n_2 \log n_2. \quad (2.3)$$

A suitable metric for measuring the size of the program, called volume, is given by

$$V = N \log_2 n \text{ bits.} \quad (2.4)$$

Intuitively, the volume is the minimum number of bits necessary to represent a complete program. The minimum possible volume that an algorithm can take is known as its potential volume, denoted by  $V^*$ . By definition,

$$V^* = (2 + n_2^*) \log_2 (2 + n_2^*), \quad (2.5)$$

where  $n_2^*$  = number of I/O parameters.

In terms of  $V$  and  $V^*$ , a metric called program level  $L$  of implementation of an algorithm can be defined as

$$L = \frac{V^*}{V}, \quad 0 < L \leq 1. \quad (2.6)$$

An approximation to this definition of  $L$ , expressed in terms of the number of operators and operands used in the program, is denoted by

$$\hat{L} = \frac{2}{n_1} * \frac{n_2}{N_2}. \quad (2.7)$$

The inverse of the program level is termed the difficulty,  $D$ . That is,

$$D = \frac{1}{\hat{L}}. \quad (2.8)$$

Therefore, as the volume of an implementation of a program increases, the program level decreases and the difficulty increases.

A metric, suggested by Halstead to characterize a programming language, is called the language level  $\lambda$  and defined as

$$\lambda = L * V^* = L^2 V. \quad (2.9)$$

Finally, a metric referred to as Effort is defined by the ratio

$$E = \frac{V}{L} = \frac{V^2}{V^*}. \quad (2.10)$$

From the definition, it is clear that the effort required to implement a



computer program increases as the size of the program increases. Therefore, since  $V^*$  is fixed for a given algorithm, software science predicts that higher level languages reduce the effort of programming.

An estimate of  $E$  can be obtained using the estimate of program level. That is,

$$\hat{E} = V/\hat{L}. \quad (2.11)$$

The effort statistic has been interpreted as a measure of the mental effort required to create a program. In other words,  $E$  represents the number of mental discriminations or decisions that a single, fluent, concentrating programmer should make in implementing the algorithm.

According to Halstead [3], the programming time should be directly proportional to the effort in a program. That is,

$$T = \frac{E}{S} \text{ Sec}, \quad (2.12)$$

where  $S$  denotes the rate of mental activity of the programmer; i.e.,  $S$  refers to the number of mental discriminations per second of which the programmer is capable. A value of  $S=18$  has been used in previous research in software science.

## 2.2 Analysis of Students' Programs

This section shows the results of the analysis of a large number of COBOL programs written by students at Ohio State University.

The data were collected by the use of a software science COBOL analyzer [2] developed by the software metrics research group at Ohio State. The analyzer, written in PL/I, counts operators and operands in the Data and Procedure divisions, and computes all software science statistics for the entire program.

The existence of this analyzer facilitates the collection of a substantial amount of data from the students of various undergraduate courses. Each student uses a simple command, called ANALYZE, to run his/her COBOL program through the analyzer. The outputs (software science statistics) of the analyzer are stored on disk. At the end of each quarter, a final report containing the software science statistics of all the students' programs is generated for analysis. The results of the analysis of all the data collected during six consecutive quarters are presented in this section. Various kinds of analysis were performed as described below.

For each particular program the following software science metrics were evaluated:

$N$ ,  $\hat{N}$ ,  $\lambda$ ,  $\hat{L}$ ,  $D$  and  $\hat{E}$ .

Each of the metric values shown in the table represents the mean of all such values for a given program during a given quarter, obtained from the set of programs written by different subjects.

Programs written by the students of two different undergraduate courses are considered in the present analysis. One course is an introduction to Data Processing (CIS 212), and the other course deals with the introduction to File Processing (CIS 313). CIS 212 is the introductory COBOL course, and CIS 313 is the next course in sequence (also using COBOL). In the first course, we examined six assignments (Lab 2, 3, 4A, 4B, 5, 6). Labs 2, 3 and 4A are of increasing complexity (in terms of size and problem concepts), and in fact, each lab is an extension of the previous lab. In other words, knowledge and understanding of Lab 2 is helpful for writing Lab 3, and knowledge of Lab 3 is somewhat directly useful to completing Lab 4A since both involve the use of a matching algorithm. Lab 4B deals with manipulating one- and two-dimensional arrays, and is conceptually different from the previous three labs. Labs 5 and 6 are of almost the same complexity, although they solve two independent

problems. Lab 5 involves sorting and Lab 6 is very similar to a report generator program. In the Fall of 1982, the curriculum was modified so that there were only five assignments instead of six. In particular, Labs 3 and 4A of the previous quarters were merged into a single assignment called Lab 3. This new Lab 3 has the identical function as old Lab 4A. All other assignments were kept unchanged.

The second course contains three assignments involving COBOL. The first program is a simple file listing program. The second assignment deals with input data validation, and the third program updates a product master file by making changes to several fields (e.g., product descriptions and prices, adding new products and deleting old products, etc.). These three programs are also of increasing complexity, although each assignment is a direct extension of the previous one.

The first set of analyses will involve the Halstead length equation and language level metrics, so that we will be interested in  $N$ ,  $\hat{N}$ , and  $\lambda$ . We will compute the mean error and mean absolute relative error in the length equation for each assignment. The mean error is defined by  $(N - \hat{N})/N$ , where  $N$  and  $\hat{N}$  represent the mean values of the  $N_i$ 's and  $\hat{N}_i$ 's, respectively, for all the subjects performing the same assignment.  $N_i$  corresponds to the value of  $N$  for  $i$ th subject. The mean absolute relative error is defined as

$$\frac{1}{n} \sum_{i=1}^n \frac{|N_i - \hat{N}_i|}{N_i},$$

where  $n$  is the number of subjects doing one particular assignment. The results of these analyses as obtained in six different quarters for the course CIS 212 are shown in the following tables.

## LAB 2 (#subjects = 19)

	N	$\hat{N}$	$N-\hat{N}/N$	Mean Abs. Rel. Err.	$\lambda$
Data Div.	643	1129	-0.75	0.67	31.3
Proc. Div.	350	519	-0.48	0.50	2.88
Program	993	1351	-0.36	0.36	2.19

## LAB 3 (#subjects = 12)

	N	$\hat{N}$	$N-\hat{N}/N$	Mean Abs. Rel. Err.	$\lambda$
Data Div.	1117	1739	-0.56	0.59	42.4
Proc. Div.	719	902	-0.25	0.28	2.34
Program	1836	2101	-0.14	0.17	1.86

## LAB 4A (# subjects = 19)

	N	$\hat{N}$	$N-\hat{N}/N$	Mean Abs. Rel. Err.	$\lambda$
Data Div.	1408	2121	-0.51	0.51	44.97
Proc. Div.	1069	1251	-0.17	0.18	1.78
Program	2477	2559	-0.03	0.07	1.30

Table 1: Spring [1981], CIS 212

LAB 4B (#subjects = 11)

	N	$\hat{N}$	$N-\hat{N}/N$	Mean Abs. Rel. Err.	$\lambda$
Data Div.	825	1365	-0.65	0.67	21.24
Proc. Div.	1213	873	+0.23	0.26	0.36
Program	2038	1836	+0.09	0.12	0.54

LAB 05 (#subjects = 11)

	N	$\hat{N}$	$N-\hat{N}/N$	Mean Abs. Rel. Err.	$\lambda$
Data Div.	1001	1559	-0.56	0.58	33.56
Proc. Div.	687	934	-0.36	0.38	1.46
Program	1688	1978	-0.17	0.20	1.13

LAB 06 (#subjects = 2)

	N	$\hat{N}$	$N-\hat{N}/N$	Mean Abs. Rel. Err.	$\lambda$
Data Div.	1011	1661	-0.64	0.67	35.86
Proc. Div.	615	941	-0.53	0.53	2.20
Program	1626	2013	-0.23	0.25	1.52

Table 2: Spring [1981], CIS 212

## LAB 2 (#subjects = 13)

	N	$\hat{N}$	$N-\hat{N}/N$	Mean Abs. Rel. Err.	$\lambda$
Data Div.	833	1317	-0.58	0.60	35.50
Proc. Div.	389	578	-0.49	0.51	2.69
Program	1222	1573	-0.29	0.30	2.06

## LAB 3 (#subjects = 22)

	N	$\hat{N}$	$N-\hat{N}/N$	Mean Abs. Rel. Err.	$\lambda$
Data Div.	1032	1583	-0.53	0.55	37.00
Proc. Div.	640	849	-0.33	0.35	2.14
Program	1659	1936	-0.17	0.18	1.54

## LAB 4A (#subjects = 9)

	N	$\hat{N}$	$N-\hat{N}/N$	Mean Abs. Rel. Err.	$\lambda$
Data Div.	1474	2310	-0.57	0.58	51.24
Proc. Div.	1050	1337	-0.27	0.28	2.04
Program	2525	2744	-0.09	0.10	1.43

Table 3: Summer [1981], CIS 212

LAB 4B (#subjects = 7)

	N	$\hat{N}$	$N-\hat{N}/N$	Mean Abs. Rel. Err.	$\lambda$
Data Div.	1233	1809	-0.47	0.52	23.90
Proc. Div.	1372	1070	+0.22	0.19	0.64
Program	2605	2258	+0.13	0.16	0.73

LAB 05 (#subjects = 14)

	N	$\hat{N}$	$N-\hat{N}/N$	Mean Abs. Rel. Err.	$\lambda$
Data Div.	1118	1818	-0.63	0.64	36.50
Proc. Div.	798	1065	-0.33	0.35	1.45
Program	1916	2282	-0.19	0.21	1.16

LAB 06 (#subjects = 10)

	N	$\hat{N}$	$N-\hat{N}/N$	Mean Abs. Rel. Err.	$\lambda$
Data Div.	1205	1872	-0.55	0.55	44.44
Proc. Div.	770	1054	-0.37	0.41	2.40
Program	1975	2218	-0.12	0.17	1.54

Table 4: Summer (1981), CIS 212

## LAB 02 (#subjects = 65)

	N	$\hat{N}$	$N-\hat{N}/N$ Mean Err.	Mean Abs. Rel. Err.	$\lambda$
Data Div.	695	1135	-0.63	0.65	29.8
Proc. Div.	336	523	-0.55	0.57	2.7
Program	1031	1412	-0.36	0.35	2.0

## LAB 03 (#subjects = 76)

	N	$\hat{N}$	$N-\hat{N}/N$ Mean Err.	Mean Abs. Rel. Err.	$\lambda$
Data Div.	1082	1652	-0.52	0.54	37.7
Proc. Div.	592	837	-0.41	0.43	2.3
Program	1674	1993	-0.19	0.20	1.7

## LAB 4A (#subjects = 70)

	N	$\hat{N}$	$N-\hat{N}/N$ Mean Err.	Mean Abs. Rel. Err.	$\lambda$
Data Div.	1429	2229	-0.55	0.57	48.1
Proc. Div.	1019	1240	-0.22	0.24	1.6
Program	2448	2674	-0.09	0.14	1.3

Table 5: Fall [1981], CIS 212



## LAB 4B (#subjects = 69)

	N	$\hat{N}$	$N-\hat{N}/N$ Mean Err.	Mean Abs. Rel. Err.	$\lambda$
Data Div.	714	1289	-0.80	0.84	26.4
Proc. Div.	1139	837	+0.26	0.24	0.45
Program	1853	1725	+0.07	0.13	0.63

## LAB 05 (#subjects = 66)

	N	$\hat{N}$	$N-\hat{N}/N$ Mean Err.	Mean Abs. Rel. Err.	$\lambda$
Data Div.	1061	1721	-0.62	0.63	40.6
Proc. Div.	720	990	-0.37	0.38	1.42
Program	1731	2150	-0.21	0.21	1.17

## LAB 06 (#subjects = 53)

	N	$\hat{N}$	$N-\hat{N}/N$ Mean Err.	Mean Abs. Rel. Err.	$\lambda$
Data Div.	1146	1607	-0.40	0.46	36.2
Proc. Div.	653	918	-0.41	0.42	2.2
Program	1799	1953	-0.08	0.12	1.4

Table 6: Fall [1981], CIS 212

## LAB 2 (#subjects = 108)

	N	$\hat{N}$	$N-\hat{N}/N$	Mean Abs. Rel. Err.	$\lambda$
Data Div.	730	1155	-0.58	0.60	28.36
Proc. Div.	353	538	-0.52	0.51	2.73
Program	1083	1398	-0.29	0.31	1.94

## LAB 3 (#subjects = 110)

	N	$\hat{N}$	$N-\hat{N}/N$	Mean Abs. Rel. Err.	$\lambda$
Data Div.	1120	1638	-0.46	0.50	35.88
Proc. Div.	616	850	-0.38	0.40	2.39
Program	1736	1968	-0.13	0.17	1.62

## LAB 4A (#subjects = 105)

	N	$\hat{N}$	$N-\hat{N}/N$	Mean Abs. Rel. Err.	$\lambda$
Data Div.	1431	2173	-0.52	0.54	46.37
Proc. Div.	1047	1226	-0.17	0.20	1.78
Program	2478	2595	-0.05	0.11	1.37

Table 7: Winter [1982], CIS 212

## LAB 4B (#subjects = 105)

	N	$\hat{N}$	$N-\hat{N}/N$	Mean Abs. Rel. Err.	$\lambda$
Data Div.	735	1272	-0.73	0.79	25.7
Proc. Div.	1151	889	+0.23	0.25	0.45
Program	1886	1712	+0.09	0.15	0.62

## LAB 05 (#subjects = 96)

	N	$\hat{N}$	$N-\hat{N}/N$	Mean Abs. Rel. Err.	$\lambda$
Data Div.	1118	1752	-0.57	0.57	42.7
Proc. Div.	742	1005	-0.35	0.37	1.7
Program	1860	2154	-0.16	0.17	1.28

## LAB 06 (#subjects = 81)

	N	$\hat{N}$	$N-\hat{N}/N$	Mean Abs. Rel. Err.	$\lambda$
Data Div.	1348	1699	-0.26	0.29	32.7
Proc. Div.	632	872	-0.38	0.43	2.29
Program	1980	2018	-0.02	0.08	1.40

Table 8: Winter [1982], CIS 212

## LAB 2 (#subjects = 154)

	N	$\hat{N}$	$N-\hat{N}/N$	Mean Abs. Rel. Err.	$\lambda$
Data Div.	720	1158	-0.61	0.60	29.14
Proc. Div.	356	552	-0.55	0.54	2.63
Program	1076	1422	-0.32	0.32	1.89

## LAB 3 (#subjects = 128)

	N	$\hat{N}$	$N-\hat{N}/N$	Mean Abs. Rel. Err.	$\lambda$
Data Div.	1111	1649	-0.43	0.49	36.78
Proc. Div.	602	850	-0.41	0.43	2.38
Program	1713	1988	-0.16	0.16	1.63

## LAB 4A (#subjects = 114)

	N	$\hat{N}$	$N-\hat{N}/N$	Mean Abs. Rel. Err.	$\lambda$
Data Div.	1470	2217	-0.51	0.51	48.08
Proc. Div.	1049	1245	-0.18	0.22	1.83
Program	2519	2638	-0.05	0.08	1.38

Table 9: Spring [1982], CIS 212

## LAB 4B (#subjects = 97)

	N	$\hat{N}$	$N-\hat{N}/N$	Mean Abs. Rel. Err.	$\lambda$
Data Div.	704	1258	-0.78	0.83	25.4
Proc. Div.	1175	845	+0.28	0.26	0.39
Program	1879	1710	+0.09	0.14	0.56

## LAB 05 (#subjects = 90)

	N	$\hat{N}$	$N-\hat{N}/N$	Mean Abs. Rel. Err.	$\lambda$
Data Div.	1114	1745	-0.56	0.55	41.3
Proc. Div.	745	1014	-0.36	0.37	1.55
Program	1859	2145	-0.15	0.16	1.17

## LAB 06 (#subjects = 73)

	N	$\hat{N}$	$N-\hat{N}/N$	Mean Abs. Rel. Err.	$\lambda$
Data Div.	1319	1638	-0.24	0.26	30.7
Proc. Div.	610	879	-0.44	0.47	2.30
Program	1929	1971	-0.02	0.07	1.29

Table 10: Spring (1982), CIS 212

LAE 2 (#subjects = 114)

	N	$\hat{N}$	$N-\hat{N}/N$	Mean Abs. Rel. Err.	$\lambda$
Data Div.	725	1140	-0.57	0.58	29.0
Proc. Div.	329	496	-0.50	0.51	2.02
Program	1054	1390	-0.32	0.32	1.73

LAE 3 (#subjects = 67)

	N	$\hat{N}$	$N-\hat{N}/N$	Mean Abs. Rel. Err.	$\lambda$
Data Div.	1349	2014	-0.49	0.50	40.0
Proc. Div.	910	1152	-0.26	0.28	1.49
Program	2259	2451	-0.08	0.11	1.13

Table 11: Fall (1982), C1S 212

## LAB 4B (#subjects = 109)

	N	$\hat{N}$	$N-\hat{N}/N$	Mean Abs. Rel. Err.	$\lambda$
Data Div.	751	1287	-0.71	0.77	23.36
Proc. Div.	1171	913	+0.22	0.20	0.37
Program	1922	1779	+0.07	0.13	0.50

## LAB 05 (#subjects = 81)

	N	$\hat{N}$	$N-\hat{N}/N$	Mean Abs. Rel. Err.	$\lambda$
Data Div.	1058	1653	-0.56	0.57	37.8
Proc. Div.	721	966	-0.33	0.34	1.23
Program	1779	2075	-0.16	0.17	1.02

## LAB 06 (#subjects = 28)

	N	$\hat{N}$	$N-\hat{N}/N$	Mean Abs. Rel. Err.	$\lambda$
Data Div.	1384	1744	-0.26	0.26	32.6
Proc. Div.	615	928	-0.50	0.53	2.17
Program	1999	2125	-0.06	0.07	1.26

Table 12: Fall [1982], CIS 212

The sign of the error indicates whether  $\hat{N}$  is an overestimate or underestimate. Note that  $\hat{N}$  is consistently an overestimate of the actual program length in all the assignments except Lab 4B. Also, neither the data nor the procedure division above yield a very acceptable length estimate. So, comparison of the results among all six quarters indicate that the length equation works well only when the data division is combined with the procedure division. In other words, the best estimate of the program length is attained when the entire program is taken into consideration. The same conclusion was drawn by Shen and Dunsmore in their software science analysis of COBOL programs [6] and was suggested by Zweben and Fung [11].

Software science postulates that the language level ( $\lambda$ ) may be used to compare various programming languages. If  $\lambda$  is indeed a property of the programming language, we might expect that it is approximately a constant for all programs written in a given language. However, the present analysis indicates that the language level is not constant. This finding agrees with that of Shen and Dunsmore [6]. It is also noticed that the  $\lambda$  for the data division is always very high compared to the  $\lambda$  of the procedure division and that of the program. This extremely large  $\lambda$  for the data division reflects the fact that COBOL provides for a compact representation of a good deal of information about type, size, structure and initial values of individual and group data items.

The other software science metrics that were evaluated for each program are NOS (number of statements), L, D and E. Each of these metrics were calculated separately for the data division, procedure division and for the entire program. For the data division, NOS denotes the number of periods, but for the procedure division NOS refers to the total number of COBOL verbs used. The primary reasons for calculating these metrics for each program are to observe how these metrics differ from one quarter to the next, and to determine if the change of D or E from one assignment to the next does really reflect the intuitive relative complexity between them.

The mean values of these metrics for each program of CIS 212 in six different quarters are presented in the following tables. The number of programs (subjects) analyzed for each lab in a given quarter is the same as that indicated in the previous set of tables.

It should be noted that the values of the software science metrics obtained in six different quarters appear to be very consistent. For example,  $N$ ,  $\hat{N}$ , Error,  $\lambda$ , NOS,  $\hat{L}$ , D and  $\hat{E}$  corresponding to each assignment are observed to be compatible from one quarter to the next. Note that there is a significant



## LAB 2

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	114	0.0821	12.17	56667
Proc. Div.	71	0.0386	25.93	73473
Program	185	0.0175	57.29	484810

## LAB 3

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	192	0.0699	14.29	133425
Proc. Div.	149	0.0264	37.88	258954
Program	341	0.0112	89.02	1418182

## LAB 4A

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	243	0.0628	15.92	185558
Proc. Div.	223	0.0149	66.85	561890
Program	466	0.0079	125.91	2699046

Table 13: Spring [1981], CIS 212

## LAB 4B

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	143	0.0589	16.96	109025
Proc. Div.	190	0.0066	152.35	1412113
Program	333	0.0058	171.88	2902761

## LAB 5

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	179	0.0683	14.65	120101
Proc. Div.	140	0.0181	55.22	300835
Program	319	0.0096	103.28	1575389

## LAB 6

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	178	0.0689	14.50	115863
Proc. Div.	118	0.0227	44.05	209213
Program	296	0.0109	91.74	1297416

Table 14: Spring [1981], CIS 212

## LAB 2

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	141	0.0764	13.08	85831
Proc. Div.	77	0.0553	18.08	85693
Program	219	0.0150	66.50	667409

## LAB 3

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	173	0.0689	14.51	120941
Proc. Div.	127	0.0217	46.06	233156
Program	300	0.0109	92.12	1286308

## LAB 4A

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	254	0.0659	15.17	187823
Proc. Div.	213	0.0160	62.33	523401
Program	467	0.0082	121.30	2655406

Table 15: Summer (1981), CIS 212

## LAB 4B

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	209	0.0513	19.48	207925
Proc. Div.	216	0.0081	123.46	1350653
Program	425	0.0060	165.86	3942305

## LAB 5

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	197	0.0652	15.34	141224
Proc. Div.	158	0.0158	63.09	395916
Program	355	0.0087	115.33	1913894

## LAB 6

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	211	0.0687	14.56	142618
Proc. Div.	153	0.0213	46.95	293731
Program	364	0.0099	100.40	1723033

Table 16: Summer (1981), CIS 212

## LAB 2

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	123	0.0782	12.8	68920
Proc. Div.	66	0.0363	27.5	64627
Program	189	0.0159	62.9	518597

## LAB 3

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	187	0.0671	14.8	128915
Proc. Div.	119	0.0236	42.3	185165
Program	306	0.0112	89.5	1253229

## LAB 4A

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	249	0.0652	15.3	183051
Proc. Div.	209	0.0145	68.8	550366
Program	458	0.0079	125.6	2670581

Table 17: Fall [1981], CIS 212

## LAB 4B

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	128	0.0714	14.0	79007
Proc. Div.	182	0.0076	130.7	1243456
Program	310	0.0067	148.6	2356694

## LAB 5

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	188	0.0701	14.2	122194
Proc. Div.	145	0.0165	60.4	327872
Program	333	0.0089	111.3	1660602

## LAB 6

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	200	0.0649	15.4	147296
Proc. Div.	126	0.0219	45.5	230030
Program	326	0.0099	100.5	1590216

Table 18: Fall [1981], CIS 212

## LAB 2

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	130	0.0742	13.5	77459
Proc. Div.	71	0.0363	27.5	68916
Program	201	0.0154	64.8	559580

## LAB 3

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	192	0.0648	15.4	142029
Proc. Div.	126	0.0236	42.3	196147
Program	318	0.0108	92.6	1368461

## LAB 4A

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	248	0.0642	15.5	186835
Proc. Div.	215	0.0151	66.3	548483
Program	463	0.0081	123.4	2677119

Table 19: Winter [1982], CIS 212

## LAB 4B

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	131	0.0703	14.22	86247
Proc. Div.	186	0.0075	132.29	1251474
Program	317	0.0065	152.28	2462322

## LAB 5

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	196	0.0701	14.25	129798
Proc. Div.	148	0.0177	56.32	317638
Program	344	0.0092	108.9	1687651

## LAB 6

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	228	0.0577	17.32	196688
Proc. Div.	123	0.0239	41.69	220578
Program	351	0.0098	101.8	1802300

Table 20: Winter [1982], CIS 212



## LAB 2

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	126	0.0745	13.40	73208
Proc. Div.	71	0.0329	29.45	72472
Program	197	0.0152	65.84	561457

## LAB 3

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	191	0.0655	15.26	135565
Proc. Div.	123	0.0236	42.30	189777
Program	314	0.0108	92.57	1314060

## LAB 4A

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	255	0.0637	15.70	192630
Proc. Div.	215	0.0152	65.78	543205
Program	470	0.0080	124.51	2702553

Table 21: Spring [1982], CIS 212

## LAB 4B

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	126	0.0708	14.12	80078
Proc. Div.	189	0.0071	139.89	1386760
Program	315	0.0064	156.95	2547522

## LAB 5

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	195	0.0691	14.47	129545
Proc. Div.	150	0.0170	58.84	340214
Program	345	0.0087	113.95	1778396

## LAB 6

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	224	0.0553	18.08	192614
Proc. Div.	118	0.0236	42.35	204212
Program	342	0.0092	108.23	1768135

Table 22: Spring [1982], CIS 212

## LAB 2

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	124	0.0738	13.5	74954
Proc. Div.	65	0.0305	32.76	73313
Program	189	0.0146	68.4	568497

## LAB 3

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	229	0.0607	16.5	182272
Proc. Div.	184	0.0146	68.15	489512
Program	413	0.0077	129.16	2494739

Table 23: Fall [1982], CIS 212

## LAB 4

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	132	0.0659	15.16	92928
Proc. Div.	192	0.0069	144.0	1476488
Program	324	0.0059	169.2	2908132

## LAB 5

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	183	0.0679	14.72	126003
Proc. Div.	145	0.0153	65.12	353810
Program	328	0.0084	118.5	1764601

## LAB 6

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	234	0.0549	18.2	200314
Proc. Div.	119	0.0225	44.36	219478
Program	353	0.0088	113.26	1921595

Table 24: Fall [1982], CIS 212

difference in the values of the software science metrics for Lab 3 in the Fall of 1982 as compared to Lab 3 in the previous quarters. In particular, the new Lab 3 has higher  $N$ ,  $\hat{N}$ , NOS,  $D$  and  $E$  values than those for the old Lab 3, but comparable to those of Lab 4A as expected due to the change made in this particular assignment in the Fall of 1982.

The final set of analyses performed on the programs of CIS 212 is the calculation of the relative values of NOS,  $D$  and  $\hat{E}$  for all the assignments to those of Lab 2. Since this is the first introductory programming course in COBOL, the purpose of these analyses is to see the change in the values of these metrics with respect to the complexity of the assignments relative to Lab 2. The explicit values of these analyses are shown separately for each program in the following tables.

## LAB 2

	NOS	Rel.NOS to Lab 2	D	Rel.D to Lab 2	$\hat{E}$	Rel. $\hat{E}$ to Lab 2
Data	114	1	12.17	1	56667	1
Proc.	71	1	25.93	1	73473	1
Prog.	185	1	57.29	1	484810	1

## LAB 3

	NOS	Rel.NOS to Lab 2	D	Rel.D to Lab 2	$\hat{E}$	Rel. $\hat{E}$ to Lab 2
Data	192	1.68	14.29	1.17	133425	2.35
Proc.	149	2.1	37.88	1.46	258954	3.52
Prog.	341	1.84	89.02	1.55	1418182	2.92

## LAB 4A

	NOS	Rel.NOS to Lab 2	D	Rel.D to Lab 2	$\hat{E}$	Rel. $\hat{E}$ to Lab 2
Data	243	2.13	15.92	1.31	185558	3.27
Proc.	223	3.14	66.85	2.58	561890	7.65
Prog.	466	2.52	125.91	2.20	2699046	5.57

Table 25: Spring [1981], CIS 212

## LAB 4B

	NOS	Rel.NOS to Lab 2	D	Rel.D to Lab 2	$\hat{E}$	Rel. $\hat{E}$ to Lab 2
Data	143	1.25	16.96	1.39	109025	1.92
Proc.	190	2.68	152.35	5.87	1412113	19.22
Prog.	333	1.8	171.88	3.00	2902761	5.99

## LAB 5

	NOS	Rel.NOS to Lab 2	D	Rel.D to Lab 2	$\hat{E}$	Rel. $\hat{E}$ to Lab 2
Data	179	1.57	14.65	1.20	120101	2.12
Proc.	140	1.97	55.22	2.13	300835	4.09
Prog.	319	1.72	103.28	1.80	1575389	3.25

## LAB 6

	NOS	Rel.NOS to Lab 2	D	Rel.D to Lab 2	$\hat{E}$	Rel. $\hat{E}$ to Lab 2
Data	178	1.56	14.50	1.19	115863	2.04
Proc.	118	1.66	44.05	1.69	209213	2.85
Prog.	296	1.6	91.74	1.60	1297416	2.68

Table 26: Spring [1981], CIS 212

## LAB 2

	NOS	Rel.NOS to Lab 2	D	Rel.D to Lab 2	$\hat{E}$	Rel. $\hat{E}$ to Lab 2
Data	141	1	13.08	1	85831	1
Proc.	77	1	18.08	1	85693	1
Prog.	218	1	66.50	1	667409	1

## LAB 3

	NOS	Rel.NOS to Lab 2	D	Rel.D to Lab 2	$\hat{E}$	Rel. $\hat{E}$ to Lab 2
Data	173	1.23	14.51	1.11	120941	1.41
Proc.	127	1.65	46.06	2.55	233156	2.72
Prog.	300	1.38	92.12	1.38	1286308	1.93

## LAB 4A

	NOS	Rel.NOS to Lab 2	D	Rel.D to Lab 2	$\hat{E}$	Rel. $\hat{E}$ to Lab 2
Data	254	1.80	15.17	1.16	187823	2.19
Proc.	213	2.77	62.33	3.45	523401	6.11
Prog.	467	2.14	121.30	1.82	2655406	3.98

Table 27: Summer [1981], CIS 212

## LAB 4B

	NOS	Rel.NOS to Lab 2	D	Rel.D to Lab 2	$\hat{E}$	Rel. $\hat{E}$ to Lab 2
Data	209	1.48	19.48	1.49	207925	2.42
Proc.	216	2.81	123.46	6.83	1350653	15.76
Prog.	425	1.95	165.86	2.49	3942805	5.91

## LAB 5

	NOS	Rel.NOS to Lab 2	D	Rel.D to Lab 2	$\hat{E}$	Rel. $\hat{E}$ to Lab 2
Data	197	1.40	15.34	1.17	141224	1.65
Proc.	158	2.05	63.09	3.49	395916	4.62
Prog.	355	1.63	115.33	1.73	1913894	2.87

## LAB 6

	NOS	Rel.NOS to Lab 2	D	Rel.D to Lab 2	$\hat{E}$	Rel. $\hat{E}$ to Lab 2
Data	211	1.50	14.56	1.11	142618	1.66
Proc.	153	1.99	46.95	2.60	293731	3.43
Prog.	364	1.67	100.40	1.51	1728033	2.59

Table 28: Summer [1981], CIS 212



## LAB 2

	NOS	Rel.NOS to Lab 2	D	Rel.D to Lab 2	$\hat{E}$	Rel. $\hat{E}$ to Lab 2
Data	123	1	12.6	1	68920	1
Proc.	66	1	27.5	1	64627	1
Prog.	189	1	62.9	1	518597	1

## LAB 3

	NOS	Rel.NOS to Lab 2	D	Rel.D to Lab 2	$\hat{E}$	Rel. $\hat{E}$ to Lab 2
Data	187	1.5	14.8	1.2	128915	1.8
Proc.	119	1.8	42.3	1.5	185165	2.8
Prog.	306	1.6	89.5	1.4	1253229	2.4

## LAB 4A

	NOS	Rel.NOS to Lab 2	D	Rel.D to Lab 2	$\hat{E}$	Rel. $\hat{E}$ to Lab 2
Data	249	2.0	15.3	1.2	183051	2.6
Proc.	209	3.2	68.8	2.5	550366	3.5
Prog.	458	2.4	125.6	1.9	2670581	5.1

Table 29: Fall (1981), CIS 212

## LAB 4E

	NOS	Rel.NOS to Lab 2	D	Rel.D to Lab 2	$\hat{E}$	Rel. $\hat{E}$ to Lab 2
Data	128	1.1	14.0	2.2	79007	1.1
Proc.	182	2.8	130.7	4.7	1243456	19.2
Prog.	310	1.6	148.6	2.4	2356694	4.5

## LAB 5

	NOS	Rel.NOS to Lab 2	D	Rel.D to Lab 2	$\hat{E}$	Rel. $\hat{E}$ to Lab 2
Data	188	1.5	14.2	1.1	122194	1.7
Proc.	145	2.2	60.4	2.2	327872	5.1
Prog.	333	1.7	111.3	1.7	1660602	3.2

## LAB 6

	NOS	Rel.NOS to Lab 2	D	Rel.D to Lab 2	$\hat{E}$	Rel. $\hat{E}$ to Lab 2
Data	200	1.6	15.4	1.2	147296	2.1
Proc.	126	1.9	45.5	1.6	230030	3.5
Prog.	326	1.7	100.5	1.5	1590216	3.1

Table 30: Fall (1981), CIS 212

## LAB 2

	NOS	Rel.NOS to Lab 2	D	Rel.D to Lab 2	$\hat{E}$	Rel. $\hat{E}$ to Lab 2
Data	130	1	13.5	1	77459	1
Proc.	71	1	27.5	1	68916	1
Prog.	201	1	64.8	1	559580	1

## LAB 3

	NOS	Rel.NOS to Lab 2	D	Rel.D to Lab 2	$\hat{E}$	Rel. $\hat{E}$ to Lab 2
Data	192	1.47	15.4	1.14	142029	1.83
Proc.	126	1.77	42.3	1.54	196147	2.84
Prog.	318	1.58	92.6	1.43	1368461	2.44

## LAB 4A

	NOS	Rel.NOS to Lab 2	D	Rel.D to Lab 2	$\hat{E}$	Rel. $\hat{E}$ to Lab 2
Data	248	1.90	15.5	1.15	186835	2.40
Proc.	215	3.02	66.3	2.4	548483	7.95
Prog.	463	2.30	123.4	1.90	2677119	4.78

Table 31: Winter [1982], CIS 212

## LAB 4B

	NOS	Rel.NOS to Lab 2	D	Rel.D to Lab 2	$\hat{E}$	Rel. $\hat{E}$ to Lab 2
Data	131	1.00	14.22	1.05	86247	1.11
Proc.	186	2.62	132.29	4.8	1251474	18.16
Prog.	317	1.57	152.28	2.35	2462322	4.40

## LAB 5

	NOS	Rel.NOS to Lab 2	D	Rel.D to Lab 2	$\hat{E}$	Rel. $\hat{E}$ to Lab 2
Data	196	1.50	14.25	1.05	129798	1.67
Proc.	148	2.08	56.32	2.05	317638	4.8
Prog.	344	1.71	108.9	1.68	1687651	3.02

## LAB 6

	NOS	Rel.NOS to Lab 2	D	Rel.D to Lab 2	$\hat{E}$	Rel. $\hat{E}$ to Lab 2
Data	228	1.75	17.32	1.28	196688	2.54
Proc.	123	1.73	41.69	1.52	220578	3.20
Prog.	351	1.75	101.8	1.57	1802300	3.22

Table 32: Winter [1982], CIS 212

## LAB 2

	NOS	Rel.NOS to Lab 2	D	Rel.D to Lab 2	$\hat{E}$	Rel. $\hat{E}$ to Lab 2
Data	126	1	13.40	1	73208	1
Proc.	71	1	29.45	1	72472	1
Prog.	197	1	65.84	1	561457	1

## LAB 3

	NOS	Rel.NOS to Lab 2	D	Rel.D to Lab 2	$\hat{E}$	Rel. $\hat{E}$ to Lab 2
Data	191	1.52	15.26	1.14	135565	1.85
Proc.	123	1.73	42.30	1.44	189777	2.62
Prog.	314	1.59	92.57	1.41	1314060	2.34

## LAB 4A

	NOS	Rel.NOS to Lab 2	D	Rel.D to Lab 2	$\hat{E}$	Rel. $\hat{E}$ to Lab 2
Data	255	2.02	15.70	1.17	192030	2.62
Proc.	215	3.03	65.78	2.23	543205	7.49
Prog.	470	2.38	124.51	1.89	2702553	4.81

Table 33: Spring [1982], CIS 212

## LAB 4B

	NOS	Rel.NOS to Lab 2	D	Rel.D to Lab 2	$\hat{E}$	Rel. $\hat{E}$ to Lab 2
Data	126	1	14.12	1.05	80078	1.09
Proc.	189	2.66	139.89	4.75	1386760	19.13
Prog.	315	1.60	156.95	2.38	2547522	4.54

## LAB 5

	NOS	Rel.NOS to Lab 2	D	Rel.D to Lab 2	$\hat{E}$	Rel. $\hat{E}$ to Lab 2
Data	195	1.55	14.47	1.08	129545	1.77
Proc.	150	2.11	58.84	1.99	340214	4.69
Prog.	345	1.75	113.95	1.73	1778396	3.16

## LAB 6

	NOS	Rel.NOS to Lab 2	D	Rel.D to Lab 2	$\hat{E}$	Rel. $\hat{E}$ to Lab 2
Data	224	1.78	18.08	1.35	192614	2.63
Proc.	118	1.66	42.35	1.43	204212	2.82
Prog.	342	1.73	108.23	1.64	1768135	3.15

Table 34: Spring [1982], CIS 212

## LAB 2

	NOS	Rel.NOS to Lab 2	D	Rel.D to Lab 2	$\hat{E}$	Rel. $\hat{E}$ to Lab 2
Data	124	1	13.5	1	74954	1
Proc.	65	1	32.76	1	73313	1
Prog.	189	1	68.4	1	568497	1

## LAB 3

	NOS	Rel.NOS to Lab 2	D	Rel.D to Lab 2	$\hat{E}$	Rel. $\hat{E}$ to Lab 2
Data	229	1.8	16.5	1.2	182272	2.4
Proc.	184	2.8	68.15	2.08	489512	6.6
Prog.	413	2.2	129.16	1.8	2494739	4.4

Table 35: Fall (1982), CIS 212

## LAB 4

	NOS	Rel.NOS to Lab 2	D	Rel.D to Lab 2	$\hat{E}$	Rel. $\hat{E}$ to Lab 2
Data	132	1.06	15.16	1.12	92928	1.2
Proc.	192	2.9	144.0	4.4	1476488	20.13
Prog.	324	1.7	169.2	2.5	2908132	5.12

## LAB 5

	NOS	Rel.NOS to Lab 2	D	Rel.D to Lab 2	$\hat{E}$	Rel. $\hat{E}$ to Lab 2
Data	183	1.5	14.72	1.09	126003	1.68
Proc.	145	2.2	65.12	1.98	358810	4.8
Prog.	328	1.7	118.5	1.7	1784601	3.13

## LAB 6

	NOS	Rel.NOS to Lab 2	D	Rel.D to Lab 2	$\hat{E}$	Rel. $\hat{E}$ to Lab 2
Data	234	1.89	18.2	1.32	200314	2.67
Proc.	119	1.83	44.36	1.35	219478	2.99
Prog.	353	1.86	113.26	1.65	1921595	3.3

Table 36: Fall [1982], CIS 212

In order to test the validity of the software science metrics, further analysis was performed on the relative difficulty of these assignments. For such analysis, the coordinator (supervisor) of this course was asked to give the approximate relative difficulties of each assignment with respect to the others. It should be mentioned that the coordinator was not aware of the analyses and research that was being performed on these assignments. He was asked to report the relative ratings of the assignments to help assess aspects of the course curriculum. The relative difficulties of the assignments reported by the coordinator can therefore be treated as an independent set of data.



In addition to the relative difficulties of the assignments, the coordinator also provided the approximate amount of time (based on his experience with the course and interaction with students who had taken the course) that a student spent completing the Lab 4A. Based on this time (in hours) of Lab 4A and the relative ratings of all the assignments (assuming the difficulty of Lab 2 to be one unit), the times needed to complete the rest of the assignments were calculated. The results, on the basis of the coordinator's report, are shown below.

	Assignment #					
	2	3	4A	4B	5	6
Relative Difficulty (coordinator)	1	2	5	6.4	3.6	3
Approximate Effort - Hours (coordinator)	7	14	35	45	25	21

The timing information thus obtained for each assignment was compared with the estimated time calculated from the software science effort metric. The range of the relative effort (assuming the effort for Lab 2 to be unity) for each program, based on the values found in six different quarters, and the corresponding range of the estimated times are shown.

	Assignment #					
	2	3	4A	4B	5	6
Relative Effort (Software Metrics)	1	1.9-2.9	4.0-5.6	5.9-6.0	2.9-3.3	2.6-2.7
Approx. Effort - Hrs. (Software Metrics)	7.5- 10.3	19.9- 21.9	41.0- 41.7	44.8- 60.9	24.3- 29.6	20.1- 26.7

These results show that software science estimate of programming times for this set of programs is somewhat consistent with the times reported by the coordinator.

For completeness of the analysis, the relative number of statements (lines of code) and the actual time allotted for each assignment are also presented.

	Assignment #					
	2	3	4A	4B	5	6
Relative # stms (Lines of Code)	1.0	1.4-1.8	2.1-2.5	1.8-1.9	1.6-1.7	1.6-1.7
Actual time allotted (weeks)	1.0	1.5	1.5	1.5	1.3	1.3

It appears, for this set of programs, that the lines of code analysis shows a very different estimate of the relative complexities than does the Halstead

effort metric. The Halstead difficulty metric also does not give the same relative complexities as does the effort metric. It is interesting to observe that the relative time allotted to these labs is inadequate no matter which relative measure is used. Traditionally, students taking this course seem to face a great deal of difficulties in completing their assignments due to lack of time. Therefore, these kinds of data analyses appear to be potentially helpful for curriculum improvements.

The analyses performed on the programs of CIS 212 were also done for the three programs of CIS 313. The results of these analyses obtained in three different quarters are shown in the following tables.

The software science metrics found for Lab 1 and Lab 2 in three different quarters appear to be quite consistent. However, in the case of Lab 3, the values of the metrics vary slightly from one quarter to the next.

It is observed that for these three programs, the length equation works equally well both for the entire program as well as for the procedure division alone. Also, note that for Lab 2 and Lab 3 (the larger labs) when the entire program is considered, the  $\hat{N}$  is an underestimate of the actual length. But for Lab 1,  $\hat{N}$  is consistently an overestimate of the actual program length. In addition, the data division analysis for each program shows a very large  $\lambda$  value, as was observed in connection with the programs of CIS 212. All other metrics, e.g., NOS, D and E, for each assignment are reasonably consistent in every quarter.

### 2.3 Analysis of University System Computer Center Programs

The analyses of the students' programs collected from the introductory COBOL courses at the Ohio State University were shown in the previous section. The purpose of this section is to study the behavior of the software science metrics for programs written in an environment which is different from the

## LAB 1 (#subjects = 23)

	N	$\hat{N}$	$N-\hat{N}/N$	Mean Abs. Rel. Err.	$\lambda$
Data Div.	758	1072	-0.41	0.41	28.16
Proc. Div.	445	669	-0.50	0.50	2.39
Program	1202	1361	-0.13	0.14	1.29

## LAB 2 (#subjects = 24)

	N	$\hat{N}$	$N-\hat{N}/N$	Mean Abs. Rel. Err.	$\lambda$
Data Div.	685	1074	-0.57	0.56	21.85
Proc. Div.	827	808	+0.02	0.12	1.37
Program	1512	1503	+0.006	0.09	1.01

## LAB 3 (#subjects = 21)

	N	$\hat{N}$	$N-\hat{N}/N$	Mean Abs. Rel. Err.	$\lambda$
Data Div.	1508	2170	-0.44	0.46	39.18
Proc. Div.	1722	1746	-0.01	0.13	1.09
Program	3230	2908	+0.09	0.10	0.73

Table 37: CIS 313, Winter 1982

## LAB 1

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	135	0.0721	13.9	79093
Proc. Div.	90	0.0282	35.5	110729
Program	225	0.0119	83.6	786455

## LAB 2

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	118	0.0671	14.9	78109
Proc. Div.	172	0.0154	65.1	402913
Program	290	0.0093	107.6	1304337

## LAB 3

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	266	0.0580	17.2	225732
Proc. Div.	382	0.0090	111.2	1603554
Program	648	0.0052	191.6	5319304

Table 38: CIS 313, Winter 1982

## LAB 1

	NOS	Rel.NOS to Lab 1	D	Rel.D to Lab 1	$\hat{E}$	Rel. $\hat{E}$ to Lab 1
Data	135	1	13.9	1	79093	1
Proc.	90	1	35.5	1	110729	1
Prog.	225	1	83.6	1	786455	1

## LAB 2

	NOS	Rel.NOS to Lab 1	D	Rel.D to Lab 1	$\hat{E}$	Rel. $\hat{E}$ to Lab 1
Data	118	0.87	14.9	1.07	78109	0.98
Proc.	172	1.9	65.1	1.8	402913	3.6
Prog.	290	1.3	107.6	1.3	1304337	1.6

## LAB 3

	NOS	Rel.NOS to Lab 1	D	Rel.D to Lab 1	$\hat{E}$	Rel. $\hat{E}$ to Lab 1
Data	266	1.9	17.2	1.2	225732	2.8
Proc.	382	4.2	111.2	3.1	1603554	14.5
Prog.	648	2.8	191.6	2.3	5319304	6.8

Table 39: CIS 313, Winter 1982

## LAB 1 (#subjects = 34)

	N	$\hat{N}$	$N-\hat{N}/N$	Mean Abs. Rel. Err.	$\lambda$
Data Div.	787	992	-0.26	0.27	24.65
Proc. Div.	381	450	-0.18	0.22	1.42
Program	1163	1245	-0.06	0.09	1.30

## LAB 2 (#subjects = 37)

	N	$\hat{N}$	$N-\hat{N}/N$	Mean Abs. Rel. Err.	$\lambda$
Data Div.	757	1136	-0.50	0.51	30.13
Proc. Div.	875	840	+0.04	0.11	1.22
Program	1632	1602	+0.02	0.07	1.06

## LAB 3 (#subjects = 31)

	N	$\hat{N}$	$N-\hat{N}/N$	Mean Abs. Rel. Err.	$\lambda$
Data Div.	1693	2370	-0.40	0.41	47.15
Proc. Div.	2121	1902	+0.10	0.11	0.85
Program	3814	3215	+0.16	0.15	0.68

Table 40: CIS 313, Spring 1982

## LAB 1

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	140	0.0664	15.06	87527
Proc. Div.	74	0.0234	42.63	108461
Program	214	0.0122	81.86	729955

## LAB 2

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	129	0.0749	13.35	78545
Proc. Div.	172	0.0140	71.25	463148
Program	301	0.0092	108.82	1434305

## LAB 3

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	296	0.0589	16.96	246934
Proc. Div.	448	0.0071	140.85	2552067
Program	744	0.0045	218.62	7476508

Table 41: CIS 313, Spring 1982



## LAE 1

	NOS	Rel.NOS to Lab 1	D	Rel.D to Lab 1	$\hat{E}$	Rel. $\hat{E}$ to Lab 1
Data	140	1	15.06	1	87627	1
Proc.	74	1	42.63	1	108461	1
Prog.	214	1	81.86	1	729955	1

## LAB 2

	NOS	Rel.NOS to Lab 1	D	Rel.D to Lab 1	$\hat{E}$	Rel. $\hat{E}$ to Lab 1
Data	129	0.92	13.35	0.88	78545	0.39
Proc.	172	2.32	71.25	1.67	463148	4.27
Prog.	301	1.41	108.82	1.33	1434305	1.96

## LAB 3

	NOS	Rel.NOS to Lab 1	D	Rel.D to Lab 1	$\hat{E}$	Rel. $\hat{E}$ to Lab 1
Data	296	2.11	16.96	1.12	246934	2.82
Proc.	448	6.05	140.85	3.30	2552067	23.52
Prog.	744	3.47	218.62	2.67	7476508	10.24

Table 42: CIS 313, Spring 1982

## LAB 1 (#subjects = 50)

	N	$\hat{N}$	$N-\hat{N}/N$	Mean Abs. Rel. Err.	$\lambda$
Data Div.	815	1088	-0.33	0.35	27
Proc. Div.	448	637	-0.42	0.43	2.2
Program	1263	1370	-0.08	0.11	1.28

## LAB 2 (#subjects = 39)

	N	$\hat{N}$	$N-\hat{N}/N$	Mean Abs. Rel. Err.	$\lambda$
Data Div.	854	1254	-0.46	0.49	26
Proc. Div.	830	850	-0.02	0.12	1.3
Program	1684	1692	-0.004	0.08	1.02

## LAB 3 (#subjects = 21)

	N	$\hat{N}$	$N-\hat{N}/N$	Mean Abs. Rel. Err.	$\lambda$
Data Div.	1334	1821	-0.36	0.39	36.5
Proc. Div.	1358	1360	-0.001	0.08	1.1
Program	2692	2435	+0.09	0.09	0.32

Table 43: CIS 313, Fall 1982

## LAB 1

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	141	0.0677	14.77	90782
Proc. Div.	89	0.0269	37.07	117566
Program	230	0.0116	86.22	862136

## LAB 2

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	145	0.0645	15.49	104038
Proc. Div.	170	0.0151	66.29	430312
Program	315	0.0088	112.58	1581638

## LAB 3

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	230	0.0604	16.5	187637
Proc. Div.	299	0.0108	92.5	1094962
Program	529	0.0062	160	3991401

Table 44: CIS 313, Fall 1982

## LAB 1

	NOS	Rel.NOS to Lab 1	D	Rel.D to Lab 1	$\hat{E}$	Rel. $\hat{E}$ to Lab 1
Data	141	1	14.77	1	90782	1
Proc.	89	1	37.07	1	117566	1
Prog.	230	1	86.22	1	862136	1

## LAB 2

	NOS	Rel.NOS to Lab 1	D	Rel.D to Lab 1	$\hat{E}$	Rel. $\hat{E}$ to Lab 1
Data	145	1.02	15.49	1.04	104038	1.14
Proc.	170	1.9	66.29	1.78	430812	3.6
Prog.	315	1.37	112.58	1.30	1581638	1.8

## LAB 3

	NOS	Rel.NOS to Lab 1	D	Rel.D to Lab 1	$\hat{E}$	Rel. $\hat{E}$ to Lab 1
Data	230	1.6	16.5	1.1	187637	2.06
Proc.	299	3.35	92.5	2.5	1094962	9.3
Prog.	529	2.3	160	1.85	3991401	4.6

Table 45: CIS 313, Fall 1982

students' environment. In order to observe such behavior, ten COBOL programs of various sizes were obtained from the University Systems Computer Center at the Ohio State University. These ten production programs, written by professional programmers, were much larger in size and perform different kinds of functions than the students' programs considered earlier.

Each of the ten programs was run through the software science analyzer at the Ohio State University to calculate the software science statistics. The results of the analyses of these programs follow.

The analyses of the University Systems programs show that for some programs

	N	$\hat{N}$	$\frac{N-\hat{N}}{N}$ (rel. err.)	$\lambda$
Data Div.	1623	3406	-1.09	30.39
Proc. Div.	2567	3162	-0.23	0.80
Program	4190	5755	-0.37	0.83

**Table 46:** Program-ID: OLD-SC-TASK 04 (File 15)

	N	$\hat{N}$	$\frac{N-\hat{N}}{N}$ (rel. err.)	$\lambda$
Data Div.	1696	3607	-1.13	32.43
Proc. Div.	2948	3506	-0.19	0.78
Program	4644	6201	-0.34	0.81

**Table 47:** Program-ID: NEW-SC-TASK 04 (File 16)

	N	$\hat{N}$	$\frac{N-\hat{N}}{N}$ (rel. err.)	$\lambda$
Data Div.	1858	3165	-0.70	31.95
Proc. Div.	1916	2315	-0.21	0.91
Program	3774	4420	-0.17	0.78

**Table 48:** Program-ID: OLL-AD-TASK 19 (File 17)

	N	$\hat{N}$	$\frac{N-\hat{N}}{N}$ (rel. err.)	$\lambda$
Data Div.	1870	3195	-0.70	32.34
Proc. Div.	2001	2397	-0.20	0.91
Program	3871	4509	-0.16	0.80

**Table 49:** Program-ID: NEW-AD-TASK 19 [File 18]

	N	$\hat{N}$	$\frac{N-\hat{N}}{N}$ (rel. err.)	$\lambda$
Data Div.	1237	2143	-0.73	33.69
Proc. Div.	2191	1924	+0.12	0.45
Program	3428	3033	+0.11	0.42

**Table 50:** Program-ID: OLD-AI-TASK 19 [File 19]

	N	$\hat{N}$	$\frac{N-\hat{N}}{N}$ (rel. err.)	$\lambda$
Data Div.	1249	2171	-0.74	34.09
Proc. Div.	1955	1949	+0.003	0.64
Program	3204	3129	+0.02	0.53

**Table 51:** Program-ID: NEW-AI-TASK 19 [File 20]

	N	$\hat{N}$	$\frac{N-\hat{N}}{N}$ (rel. err.)	$\lambda$
Data Div.	956	1968	-1.05	35.33
Proc. Div.	2669	2830	-0.06	0.90
Program	3625	4136	-0.14	0.78

**Table 52:** Program-ID: OLD-YT-TASK 42 [File 21]

	N	$\hat{N}$	$\frac{N-\hat{N}}{N}$ (rel. err.)	$\lambda$
Data Div.	1215	2497	-1.05	37.80
Proc. Div.	3127	3210	-0.02	0.88
Program	4342	4821	-0.11	0.80

**Table 53:** Program-ID: NEW-YT-TASK 42 [File 22]

	N	$\hat{N}$	$\frac{N-\hat{N}}{N}$ (rel. err.)	$\lambda$
Data Div.	1782	3901	-1.19	61.00
Proc. Div.	5133	4680	+0.08	1.51
Program	6915	7204	-0.04	1.47

**Table 54:** Program-ID: OLD-VX-TASK 75 [File 23]

	N	$\hat{N}$	$\frac{N-\hat{N}}{N}$ (rel. err.)	$\lambda$
Data Div.	2242	4755	-1.12	61.02
Proc. Div.	6835	5642	+0.17	1.23
Program	9077	8552	+0.05	1.22

**Table 55:** Program-ID: NEW-VX-TASK 75 [File 24]

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	268	0.0465	21.5	302322
Proc. Div.	590	0.0060	166.6	3704166
Program	858	0.0046	217.4	8537391

**Table 56:** Program-ID: OLD-SC-TASK 04 [File 15]

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	278	0.0468	21.4	316410
Proc. Div.	653	0.0055	181.8	4706727
Program	931	0.0043	232.5	10220465

**Table 57:** Program-ID: NEW-SC-TASK 04 [File 16]



	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	322	0.0448	22.3	355401
Proc. Div.	385	0.0076	131.6	2087500
Program	707	0.0048	208.3	7113125

**Table 58:** Program-ID: OLD-AD-TASK 19 [File 17]

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	324	0.0449	22.3	357371
Proc. Div.	401	0.0074	235.1	2250270
Program	725	0.0048	208.3	7315625

**Table 59:** Program-ID: NEW-AD-TASK 19 [File 18]

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	208	0.0580	17.2	172706
Proc. Div.	501	0.0051	196.1	3467450
Program	709	0.0038	263.2	7764475

**Table 60:** Program-ID: OLD-AI-TASK 19 [File 19]

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	211	0.0580	17.2	174724
Proc. Div.	437	0.0064	156.3	2468750
Program	648	0.0044	227.3	6291590

**Table 61:** Program-ID: NEW-AI-TASK 19 [File 20]

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	162	0.0680	14.7	112382
Proc. Div.	642	0.0063	158.7	3609206
Program	804	0.0049	204.1	6636938

**Table 62:** Program-ID: OLD-YT-TASK 42 [File 21]

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	209	0.0613	16.3	164127
Proc. Div.	718	0.0057	175.4	4755614
Program	927	0.0045	222.2	8833333

**Table 63:** Program-ID: NEW-YT-TASK 42 [File 22]

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	326	0.0623	16.1	252279
Proc. Div.	1164	0.0057	175.4	8202456
Program	1490	0.0047	212.7	14170000

**Table 64:** Program-ID: OLD-VX-TASK 75 [File 23]

	NOS	$\hat{L}$	$D=1/\hat{L}$	$\hat{E}$
Data Div.	412	0.0548	18.2	370802
Proc. Div.	1503	0.0044	227.3	14505000
Program	1915	0.0037	270.3	24149459

**Table 65:** Program-ID: NEW-VX-TASK 75 [File 24]

the length equation works better for the procedure division alone, and for others the length equation gives a better estimate when the entire program is considered (i.e., when the data division is combined with the procedure division). It should be noted, however, that for almost all of these programs both the procedure division and the entire program give reasonable values of the error. A similar conclusion was drawn by Shen and Dunsmore from the analysis of their COBOL analyzer program itself [6]. Since the data division is a significant part of any COBOL program and may require a considerable amount of programming effort, it still seems reasonable to include it in software science studies.

For each of these production programs, the  $\lambda$  value for the data division is observed to be much higher than that for the procedure division or the whole program, as was observed in connection with the analyses of the students' programs. It is also interesting to note that the values of  $\lambda$  for these programs are generally much lower than the  $\lambda$  values for the student programs. Shen and Dunsmore observed that  $\lambda$  seems to fall as the program size increases. However, the two largest programs in this sample have the largest values of  $\lambda$ !

Additional metrics, e.g., NOS,  $\hat{L}$ , D and  $\hat{E}$  have also been evaluated for each program. It is interesting to note that for this particular set of programs, the NOS and  $\hat{E}$  metrics order the programs in the same way. However, we were unable to obtain data from University Systems which would allow us to validate  $\hat{E}$  as an estimate of actual development effort.

#### 2.4 Comparison of the Results Between the OSU Analyzer and the Purdue Analyzer

In order to find out the differences between the software science metrics values when using two different counting strategies, the programs collected from University Systems were run through two different COBOL analyzers. One COBOL analyzer was developed at OSU, and the other analyzer was produced by the software metrics research group at Purdue University. The metrics values produced by the two analyzers are shown in the following tables. Since it was observed that the best software science estimates are generally achieved for the entire program rather than for the data or procedure division alone, this section includes the results only of the analysis of the entire program (i.e., combination of data and procedure division). The differences noticed in the values of the metrics are due to the differences in counting strategies of operators and operands as proposed by the two groups [2]. It appears that the Purdue analyzer was unable to analyze the largest program (File 24) among all these ten programs obtained from the University Systems.

The results of both the analyzers on the University System production programs show that the length equation works well for almost all of the programs in the set. When the OSU analyzer is used, it is noticed that the length equation produces positive error for the two smallest and the largest program. For all other programs, the length equation produces negative errors. On the other hand, the use of Purdue's analyzer shows that the length equation produces negative errors for all the programs in the set except the smallest program. These results, as obtained for this set of ten production programs, contradicts the result observed for 11 AIRMICS production programs

PROG. #	PURDUE				OSU			
	ETA1	N1	ETA2	N2	ETA1	N1	ETA2	N2
File 15	101	1486	329	1368	123	2286	540	1904
File 16	105	1633	360	1562	127	2531	579	2113
File 17	87	1324	268	1212	98	1970	431	1804
File 18	88	1364	276	1250	99	2026	439	1845
File 19	93	1303	200	1028	102	1958	288	1470
File 20	87	1198	208	962	96	1800	303	1404
File 21	87	1118	276	1281	103	2073	399	1552
File 22	91	1323	326	1532	107	2442	463	1900
File 23	89	2385	499	2243	100	3982	693	2933
File 24	-	-	-	-	113	5259	806	3818

PROG. #	PURDUE				OSU			
	N	$\hat{N}$	$\frac{N-\hat{N}}{N}$	$\lambda$	N	$\hat{N}$	$\frac{N-\hat{N}}{N}$	$\lambda$
File 15	2854	3423	-0.20	0.55	4190	5755	-0.37	0.83
File 16	3195	3761	-0.18	0.55	4644	6201	-0.34	0.81
File 17	2536	2722	-0.07	0.56	3774	4420	-0.17	0.78
File 18	2614	2806	-0.07	0.55	3871	4509	-0.16	0.80
File 19	2331	2137	+0.08	0.34	3428	3033	+0.11	0.42
File 20	2160	2162	-0.001	0.44	3204	3129	+0.02	0.53
File 21	2399	2798	-0.17	0.50	3625	4136	-0.14	0.78
File 22	2855	3313	-0.16	0.53	4342	4821	-0.11	0.80
File 23	4628	5048	-0.09	1.06	6915	7204	-0.04	1.47
File 24	-	-	-	-	9077	8552	+0.05	1.22

PROG. #	PURDUE		OSU	
	$\hat{E}$	Estimated Time in Hrs.	$\hat{E}$	Estimated Time in Hrs.
File 15	5201458	80	8537391	131
File 16	6434318	99	10220465	158
File 17	4212549	65	7113125	110
File 18	4447800	68	7315625	113
File 19	4548095	70	7764473	120
File 20	3578873	55	6291590	97
File 21	4121212	63	6636938	102
File 22	5287021	82	8833333	136
File 23	8532264	132	14170000	219
File 24	-	-	24149459	373

reported by Shen and Dunsmore [6], namely that the length equation produces negative errors for small programs but positive errors for large programs.

It was also found [6, 7] that the range of program sizes for which the length equation appears to work best is  $2000 \leq N < 4000$ . It is of interest to note that the program length prediction for all the programs in that range are quite satisfactory.

Another result reported [6] for AIRMICS production programs is that language level ( $\lambda$ ) is affected by then size of the program. In particular, large  $N$ 's are accompanied by smaller  $\lambda$ 's. However, the  $\lambda$  values obtained for these 10 production programs, using two different analyzers, do not seem to support this particular result.

In summary, the COBOL studies provided mixed results. On the positive side, the length estimate was once again found generally satisfactory. The effort measure also provided some more evidence that it can be used to approximate development time, and can reliably estimate relative effort of development, at least for (small) student programs. Use of this information to assist in curriculum control was also suggested.

On the negative side, further evidence against the utility of the language level measure was obtained. Large variances were observed, consistent with other studies, and conflicting evidence of the relationship between  $\lambda$  and  $N$  to that of Shen and Dunsmore was obtained. Contrary results to those of previous authors concerning the sign of the relative error in the length estimate was also obtained. The counting strategies for the OSU and Purdue analyzers appears sufficiently different that the actual values of several of the metrics changes dramatically. This has very serious implications if these metrics are to be used in an absolute sense, say as estimates of development time. It once again points out the need for taking great care in interpreting the results of software science studies, and in comparing these results with those of other researchers.



### 3. Relationships Among Various Software Metrics

#### 3.1 Motivation

Software complexity metrics appear to have numerous advantages in the design, construction and maintenance of software systems. While several such metrics have been defined, and some of them have been validated on actual systems, significant work remains to be done to establish the relationships among these metrics. This chapter shows the relationships among four different complexity metrics, which were calculated for a small set of COBOL programs. The primary motivation of this study is to investigate the extent to which each of the four complexity metrics correctly orders the programs by their actual programming time, the hypothesis being that a more complex program takes longer to write. The metrics considered in this study were Halstead's software science effort measure [3], McCabe's cyclomatic complexity metric [5], Henry and Kafura's information flow complexity metric [4], and Davis's chunk model complexity metric [1]. Since the information flow complexity is primarily used to define the complexity of an individual procedure rather than the complexity of the entire program, a section has been included to find the relationship between the Halstead's effort and the information flow complexity for each module using the same set of COBOL programs.

#### 3.2 Background and Definitions of Metrics

The definitions of each complexity metric considered in this study are given in this section.

The Halstead's Effort ( $E$ ) is computed using the equation (2.11), as defined in Chapter 2.

### McCabe's Cyclomatic Complexity

The cyclomatic number,  $V(G)$ , of a graph with  $e$  edges, and  $p$  connected components is [5]

$$V(G) = e - n + p.$$

McCabe assumes that every program can be represented by a directed graph where the edges represent different control paths and the nodes represent processing segments. The number of components,  $p$ , can be identified with the number of different routines in a program. It can be shown that the cyclomatic complexity for a program is a function of the number of predicates in the program. Formally, if  $G$  is a program containing  $M$  binary decision points (e.g., IF, WHILE, FOR) then the cyclomatic complexity,  $V(G)$ , is

$$V(G) = M + 1.$$

McCabe has shown that this value of  $V(G)$  denotes the cardinality of a basis set of paths through a program.

### Henry and Kafura's Information Flow Complexity Metric (IFC)

The information flow complexity metric [4] deals directly with system connectivity by observing the flow of information or control among system components. In this case, the formula for defining the complexity value of a procedure is

$$IFC = Length * (fan-in * fan-out)^2,$$

where

Length of a procedure is defined as the number of lines of text in the source code for the procedure.

Fan-in of a procedure  $A$  is the number of local flows into procedure  $A$  plus the number of data structures from which the procedure  $A$  retrieves information.

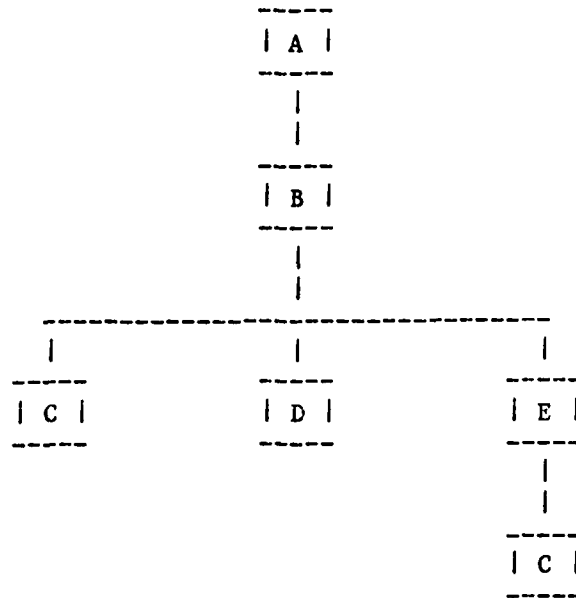
Fan-out of a procedure  $A$  is the number of local flows from  $A$  plus the number of data structures which procedure  $A$  updates.

The term "local flow" is defined as follows:

There is a local flow of information from module A to module B if one or more of the following conditions holds.

1. If A calls B,
2. If B calls A and A returns a value to B, which B subsequently utilizes, or
3. If e calls A and B passing an output value from A to B.

For example, consider the following structure chart for a COBOL program:



- A.  
PERFORM B.  
ADD 1 TO X.
- B.  
PERFORM C.  
ADD P, Q GIVING X.  
MOVE Y TO Z.  
PERFORM D.  
PERFORM E.  
MOVE ZERO TO Q.
- C.  
.  
.  
.  
MOVE 5 TO P.  
ADD 1 TO Q.
- D.  
.  
.  
.  
ADD Z TO K.
- E.  
.  
.  
.  
PERFORM C.

In this particular example,

$$\begin{aligned}\text{Fan-in of B} &= (\# \text{ local flow into B}) + (\# \text{ DS from which B retrieves information}) \\ &= 2 + 3 = 5\end{aligned}$$

since, there is a local flow from A to B (by definition 1) and also from C to B (by definition 2). The data elements from which B retrieves information are P, Q and Y.

$$\begin{aligned}\text{Fan-out of B} &= (\# \text{ local flow from B}) + (\# \text{ DS which B updates}) \\ &= 3 + 3 = 6\end{aligned}$$

since, from B there is local flow to C, D and E, and the data elements updated by B are X, Z and Q.

The details of the counting strategy used to find information flow complexity for COBOL programs are explicitly listed in Appendix A.

#### Chunk Model Complexity (C)

In this approach, chunks are used as a basis of complexity measurement. The original idea is based on the fact that an expert programmer does not understand a program on a character by character or line by line basis. Rather programmers assimilate groups of statements which have a common function. These groups are called "chunks". Therefore, the idea is to consider a program as divided into more than one chunks based on some definite criteria (which vary from language to language). The complexity of each chunk is determined, and these can then be added up to calculate the complexity of the program. For example, in a COBOL program each performed paragraph can be treated as a chunk.

The final formula for program complexity can then be written in the form [1]:

$$C = \sum_{i=1}^{\text{\# chunks}} \left( \sum_{m=0}^{f_i} C_i * R^m \right)$$

$$= \sum_{i=1}^{\text{\# chunks}} \left( \sum_{m=0}^{f_i} R^m \right)$$

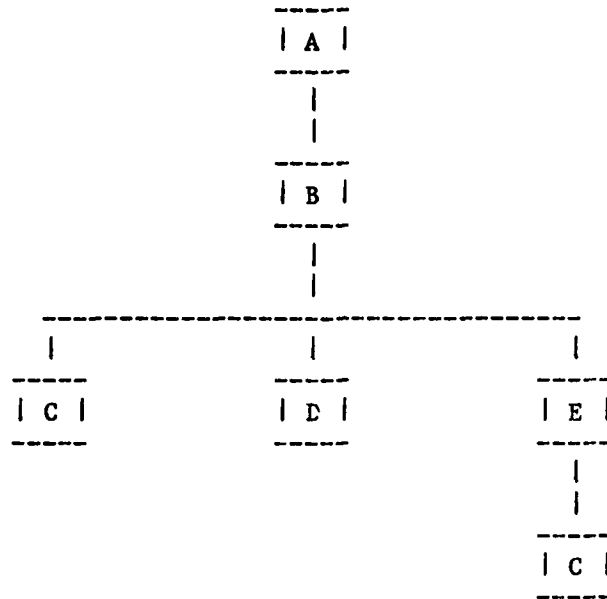
where

$C_i$  = Complexity of the  $i$ th chunk (e.g., lines of code)  
 $f_i$  = fan-in for the  $i$ th chunk  
 $R$  = 2/3, Review Constant.

It should be noted that in this case the definition of fan-in is different from the fan-in used in connection with information flow complexity of the previous section. In particular, here the term "fan-in" accounts for the number of other chunks affected by a particular chunk. Formally, chunk A is affected by chunk B, denoted by  $A \Rightarrow B$ , if any one of the following conditions is true.

1. Chunk A has a control connection to B:  $A \Rightarrow cB$ ,
2. Chunk A has a data connection to B:  $A \Rightarrow dB$ .

Now, A has control connection to B, if A contains a PERFORM or GO TO statement which references B. On the other hand, A has data connection to B if there is some variable X whose value is changed in B and referenced in A. As an example, consider the same structure chart as before:



- A.  
PERFORM B.  
ADD 1 TO X.
- B.  
PERFORM C.  
ADD P, Q GIVING X.  
MOVE Y TO Z.  
PERFORM D.  
PERFORM E.  
MOVE ZERO TO Q.
- C.  
.  
.  
.  
MOVE S TO P.  
MOVE 1 TO Q.
- D.  
.  
.  
.  
ADD Z TO K.
- E.  
.  
.  
.  
PERFORM C.

In this case,

Fan-in of B = (# control connection to B) +  
                   (# data connection to B)  
               = 1 + 1, (A => cB, A => dB), D => dB  
               = 2.

Note that, although A has both control and data connection to B, it is counted only once in calculating the fan-in of B. Furthermore, it should be mentioned that the fan-in of chunk A will be entirely determined by the number of data connections to A, since there is no control connection to A.

### 3.3 Source of Data and the Comparison of the Metrics

The COBOL programs considered in this study were written by the students of an undergraduate course on "introduction to file processing" at the Ohio State University. Two different sets of programs were analyzed. The first set of programs deals with input data validation (i.e., the program edits transactions for a master file update). The other set of programs is concerned with product master file update; that is, the program updates a product master file by making changes to several fields, e.g., product description and prices, adding new products and deleting old products. It should be mentioned that the students in this course are fairly familiar with COBOL syntax, because this is their second course using COBOL. Each student, while visiting a program, is also required to keep track of the program development history using a shot log. This shot log provides detailed information about actual programming time needed in various activities (e.g., time in designing, in coding and in modifying at each subsequent run, etc.). About 20 students took part in the study. However, after all the shot logs were obtained, only three subjects corresponding to extreme situations were selected. In other words, for each type of program, the study was made using only a sample subject which showed small, medium and very high development times respectively. For each of the programs considered in the analysis, all four complexity metrics were calculated.



In order to find Halstead's effort, each program was run through the Software Science Analyzer [2] developed by the Software Metrics Research group at the Ohio State University. All other complexity metrics were evaluated manually using the source code and the hierarchy chart for each program. The detailed derivations of these results are included in Appendix A.

The results of calculating the four different metrics for two sets of COBOL programs are summarized below:

Prog. Set	Program ID	Reported Time in Hrs.	Halstead Effort (E)	V(G)	Information Flow Complexity (IFC)	Chunk Model Complexity (C)
1	CIS 313-L2-TC0650	15	1104705	36	5051929	314
	CIS 313-L2-TC0671	10.5	1241195	41	8661844	295
	CIS 313-L2-TC0645	27	1267142	43	4402984	362
2	CIS 313-L3-TC0645	31	6895476	72	25709599	1031
	CIS 313-L3-TC0671	40	5503529	72	23343021	860
	CIS 313-L3-TC0622	49	6938627	71	36751009	1309

It is observed that for the first set of programs, the chunk model complexity shows very good agreement with the reported time in the sense that

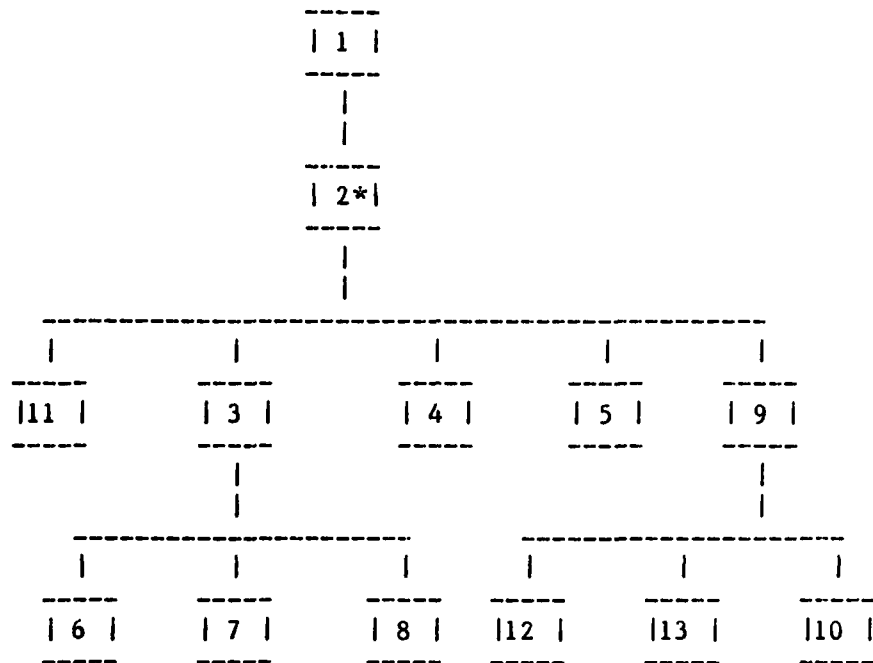
it orders the three programs correctly. On the other hand, for this set the information flow complexity seems to be the worst predictor of time. Finally, there is a good agreement between the complexity values of Halstead E and V(G). For the second set of programs, E, information flow, and the chunk model metrics order the programs identically. However, none of these metrics orders the three programs properly with respect to reported time by the programmers.

#### 3.4 Module Based Comparison Between Effort and Information Flow Complexity

It should be noted that the complexity of a procedure depends on two factors, namely, the complexity of the procedure code and the complexity of the procedure's connections to its environments. Halstead and McCabe's complexity measures appear to use only the first factor. However, the information flow complexity measure, while using both the factors to some degree, concentrates primarily on the procedure's connections to its environment through the fan-in and fan-out. Since both the Halstead measure and the information flow complexity measure fail to give uniform weight to both the factors mentioned above, an attempt was made to find out how these two measures differ with respect to individual modules (performed paragraphs) for the same sets of COBOL programs used in the previous section. In other words, the purpose is to see if both the measures behave the same way, or to find out how they differ. The detailed results of the module-based comparison for each sample are presented in this section. Note that E for each module was obtained by running each procedure division paragraph (along with its associated data division entries) separately through the Software Science Analyzer [2].

Consider first the program C1S 313-L2-TC0645. The hierarchy chart for this program is shown below, where each box denotes a module in the program. The asterisk (\*) in module 2 indicates that it is an iterated part of module 1. It is observed that for most of the modules there is very high correlation

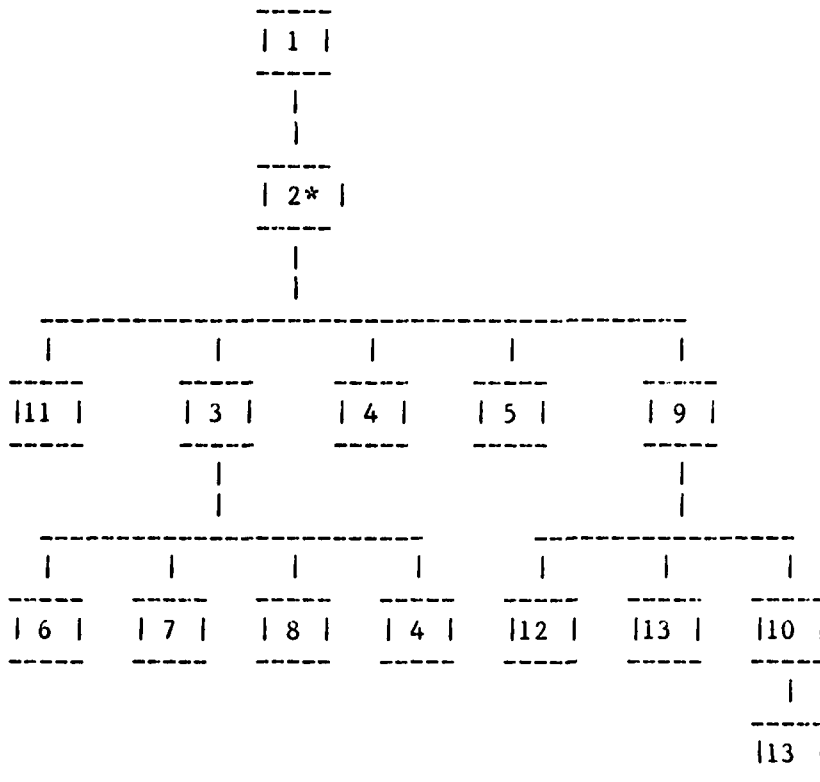
between the complexity values in the two measures. For example, module 2 containing the only major loop in the program shows highest complexity in both the measures. Modules 6 and 7 seem to be equally complex, as expected from the code and design shown below. Finally, complexity values in both the measures show that modules 11, 12 and 13 are equally complex, as expected.



Module #	Halstead, E	IFC
1	31325	23750
2	102441	3630000
3	17509	3000
4	24823	5904
5	49232	336960
6	24962	10496
7	25204	10496
8	42094	22000
9	29456	158760
10	56099	88218
11	26742	37800
12	26742	37800
13	26742	37800

**Table 66:** Program 1D: CIS 313-L2-TC0645

Now consider the program CIS 313-L2-TC0671. The hierarchy chart for this program is shown below.



The numerical values of the Effort metric and the information flow

## \*MODULE 6

TYPE-1-4.

\*\*\*\*\*

\* THIS PARAGRAPH CHECKS TO MAKE SURE BOTH THE UPDATE  
 \* PRICE AND THE DESCRIPTION ARE NON-BLANK, IF NOT  
 \* A FLAG IS SET.

\*\*\*\*\*

IF PR-PRICE IS NOT NUMERIC OR

PR-DESCRIPTION OF PUF-REC IS = ' '

MOVE 1 TO FLAG1

MOVE 1 TO FLAG7

MOVE PR-PRICE TO PRICE.

IF PR-PRICE IS NUMERIC

MOVE PR-PRICE1 TO PRICE1.

## \*MODULE 7

TYPE-2-6-9.

\*\*\*\*\*

\* FOR ACTIVATE, DEACTIVATE AND DELETE BOTH THE UPDATE  
 \* PRICE AND DESCRIPTION MUST BE BLANK; THIS PARAGRAPH  
 \* SETS A FLAG IF THEY ARE NOT.

\*\*\*\*\*

IF PR-PRICE IS NOT = ' ' OR

PR-DESCRIPTION OF PUF-REC NOT = ' '

MOVE 1 TO FLAG1

MOVE 1 TO FLAG10

IF PR-PRICE IS NUMERIC

MOVE PR-PRICE1 TO PRICE1.

MOVE PR-PRICE TO PRICE.

## \*MODULE 11

## HEADER-PR1.

```
*****
* THIS IS THE PARAGRAPH TO PRINT A HEADING ON THE TOP
* OF THE PAGE FOR ALL THE UPDATE REQUESTS.
*****
WRITE PRINTER-REC1 FROM T-HDG
  AFTER ADVANCING TO-TOP-OF-PAGE.
```

```
MOVE 'ALL UPDATE REQUESTS' TO TITLE-PHRASE.
ADD 1 TO PAGE-CT1.
MOVE PAGE-CT1 TO PAGE-OUT.
WRITE PRINTER-REC1 FROM T-HDG-1
  AFTER ADVANCING 2 LINES.
WRITE PRINTER-REC1 FROM HDG-1
  AFTER ADVANCING 2 LINES.
WRITE PRINTER-REC1 FROM HDG-2
  AFTER ADVANCING 1 LINES.
MOVE 6 TO LINE-CT1.
```

## \*MODULE 12

## HEADER-PR2.

```
*****
* THIS PARAGRAPH PRINTS A HEADING ON THE TOP OF THE PAGE
* FOR THE REPORT OF ALL THE GOOD UPDATE REQUESTS.
*****
WRITE PRINTER-REC2 FROM T-HDG
  AFTER ADVANCING TO-TOP-OF-PAGE.
MOVE 'GOOD UPDATE REQUESTS' TO TITLE-PHRASE.
ADD 1 TO PAGE-CT2.
MOVE PAGE-CT2 TO PAGE-OUT.
WRITE PRINTER-REC2 FROM T-HDG-1
  AFTER ADVANCING 2 LINES.
WRITE PRINTER-REC2 FROM HDG-1
  AFTER ADVANCING 2 LINES.
WRITE PRINTER-REC2 FROM HDG-2
  AFTER ADVANCING 1 LINES.
MOVE 6 TO LINE-CT2.
```

## \*MODULE 13

## HEADER-PR3.

```
*****
* THIS IS THE PARAGRAPH TO PRINT A HEADING ON THE TOP
* OF THE PAGE FOR ALL THE INVALID UPDATE REQUESTS.
*****
WRITE PRINTER-REC3 FROM T-HDG
  AFTER ADVANCING TO-TOP-OF-PAGE.
MOVE 'INVALID UPDATE REQUESTS' TO TITLE-PHRASE.
ADD 1 TO PAGE-CT3.
MOVE PAGE-CT3 TO PAGE-OUT.
WRITE PRINTER-REC3 FROM T-HDG-1
  AFTER ADVANCING 2 LINES.
WRITE PRINTER-REC3 FROM HDG-1
  AFTER ADVANCING 2 LINES.
WRITE PRINTER-REC3 FROM HDG-2
  AFTER ADVANCING 1 LINES.
MOVE 6 TO LINE-CT3.
```

complexity metric for each module are listed in the following table.

Module #	Halstead E	IFC
1	39654	25625
2	114327	7978176
3	19090	4752
4	11349	864
5	16414	7936
6	14203	2187
7	11769	864
8	19257	4320
9	21307	112896
10	81104	348160
11	30401	54432
12	29491	54432
13	29491	67200

**Table 67:** Program ID: CIS 313-L2-TC0671

Note that, for most of the modules, the complexity measures in the two cases are comparable. For example, module 2 is the only major loop in the program, and the complexity value of this module is highest for both the measures as might be expected. Modules 4 and 7 are very small and perform very similar functions (in fact, both the modules contain only one IF statement and a MOVE statement). The complexity values of these two modules are very close for both measures. However, the difference in the complexity values between modules 12 and 13 is noticeable. In particular, the E measures of modules 12 and 13 are exactly the same, but the information flow complexity suggests that the complexity of module 13 is somewhat higher than that of module 12. Examining the actual code (as shown below), it can be seen that both of these modules have virtually identical functions and contain the same number of lines of code. So, intuitively, the complexity should be about the same.

The reason for the variation of information flow complexity values between modules 12 and 13 is due to the higher fan-in value of module 13. The inconsistency in the two complexity metrics suggests that the two metrics serve different purposes. In particular,

## \*MODULE 12

```
*****
*THE NEW-HEADER-2 IS USED TO PRINT THE NEW *
*HEADER AT THE TOP OF A NEW PAGE FOR      *
*PRINTER-FILE-2.                          *
*****
```

## NEW-HEADER-2.

```
IF TOTAL-LINES-VALID NOT LESS THAN MAX-LINES
  MOVE PAGE-COUNT-VALID TO VALID-UPDATE-PAGE
  WRITE PRINTER-RECORD-2 FROM VALID-UPDATE-HEADER
  AFTER ADVANCING TO-TOP-OF-PAGE
  WRITE PRINTER-RECORD-2 FROM DATE-HEADER
  AFTER ADVANCING 1 LINES
  WRITE PRINTER-RECORD-2 FROM TOPPER-1
  AFTER ADVANCING 3 LINES
  WRITE PRINTER-RECORD-2 FROM TOPPER-2
  AFTER ADVANCING 1 LINES
  MOVE 6 TO TOTAL-LINES-VALID
  ADD 1 TO PAGE-COUNT-VALID.
```

## \*MODULE 13

```
*****
*THE NEW-HEADER-3 IS USED TO PRINT THE NEW *
*HEADER AT THE TOP OF A NEW PAGE FOR      *
*PRINTER-FILE-3.                          *
*****
```

## NEW-HEADER-3.

```
IF TOTAL-LINES-INVALID NOT LESS THAN MAX-LINES
  MOVE PAGE-COUNT-INVALID TO INVALID-UPDATE-PAGE
  WRITE PRINTER-RECORD-3 FROM INVALID-UPDATE-HEADER
  AFTER ADVANCING TO-TOP-OF-PAGE
  WRITE PRINTER-RECORD-3 FROM DATE-HEADER
  AFTER ADVANCING 1 LINES
  WRITE PRINTER-RECORD-3 FROM TOPPER-1
  AFTER ADVANCING 3 LINES
  WRITE PRINTER-RECORD-3 FROM TOPPER-2
  AFTER ADVANCING 1 LINES
  MOVE 6 TO TOTAL-LINES-INVALID
  ADD 1 TO PAGE-COUNT-INVALID.
```



1. Effort may be a development-oriented metric once the specification is given rather than a maintenance-oriented metric. In contrast, the information flow complexity may be a better indicator of modifiability, since information flow complexity strongly emphasizes the procedure's connection to its environment through fan-in and fan-out. (The chunk model might also be a better indicator of modifiability.)
2. Effort may be a suitable metric for understanding a subsystem in terms of its particular function and not in terms of understanding a modification to that subsystem in connection with the whole system. That is, Effort is not concerned with the interconnectivity or interdependency of modules with one another.
3. Effort helps to understand a module in isolation, i.e., one subactivity of the system, not the total activity. But the information flow complexity metric accounts for the total activity required for maintenance purposes.

Therefore, a conclusion that can be drawn is that the Halstead Effort measure may not be sensitive to the integration problem, in contrast to the information flow complexity metric. Similar discrepancies among the modules in other programs of the set can also be justified in the same way. The next chapter will further explore the problem of measuring integration effort.

#### 4. Development of a New Approach to Measuring Software Effort

##### 4.1 Motivation

In this chapter, an attempt is made to develop a new approach for measuring the software effort in a COBOL environment, where effort will be based on programming time.

The programming time may be estimated using the software science formulas (2.11) - (2.12), as discussed in Chapter 2. The experiments reported by Halstead [3] showing the comparison of actual programming times and the estimated programming times using  $E$  involved only one subject. Another small experiment conducted later shows that when the effort measure is applied to large programs with multiple modules, it consistently overestimates programming time [8]. A more recent set of experiments suggests that larger modules in multi-module programs should be conceptually broken into smaller parts before applying the  $E$  measure [9, 10]. It is observed that the use of  $S = 18$  to convert the  $E$  measure to  $T$  works best for modules which take less than two hours to produce and which are less than 50 lines of code in length.

Our study has been conducted in a COBOL environment in order to compare the actual programming time and the estimated programming time. Two different sets of COBOL programs were considered for this study (the detailed descriptions of these programs were given in the previous chapter). Each set contains three subjects.

After these programs were submitted by the programmers, each program was run through the Software Science Analyzer [2] at the Ohio State University in order to get the various software science metrics, including Effort. The effort value for each program along with the estimated time (calculated by using the equation (2.12) and the reported time provided by the programmer are shown below. In the present analysis,  $S$  is considered to be 18, consistent



AD-A141 467

A STUDY OF THE APPLICATION OF SOFTWARE METRICS TO COBOL  
(U) OHIO STATE UNIV COLUMBUS COMPUTER AND INFORMATION  
SCIENCE RES. N C DEBNATH ET AL JUL 83

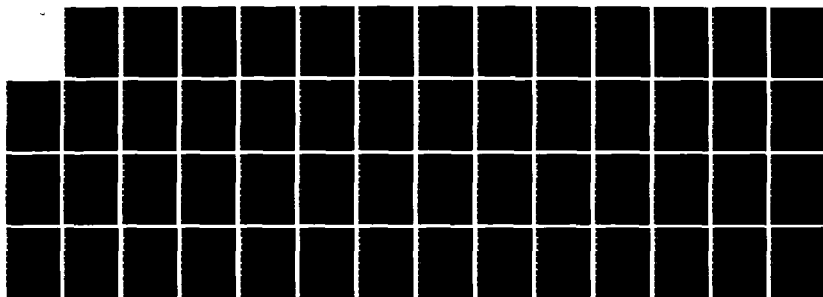
2/2

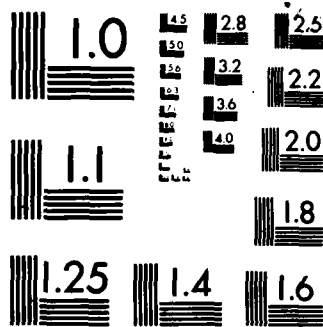
UNCLASSIFIED

OSU-CISRC-TR-84-3 ARO-17150.5-EL

F/G 9/2

NL





MICROCOPY RESOLUTION TEST CHART  
NATIONAL BUREAU OF STANDARDS-1963-A

with other studies which have examined Halstead's programming time relationship [3].

Prog. Set	Program-ID	EFFORT, E	Estimated Time, T, in Hours	Reported Time in Hours
1	CIS 313-L2-TC0650	1104705	17	15
	CIS 313-L2-TC0671	1241195	19	10.5
	CIS 313-L2-TC0645	1267142	19.5	27
2	CIS 313-L3-TC0671	5503529	85	40
	CIS 313-L3-TC0645	6895476	106	31
	CIS 313-L3-TC0622	6938627	107	49

It should be mentioned that the programs in Set 1 contain 13 to 14 paragraphs in the procedure division while the programs in Set 2 contain between 32 and 36 procedure division paragraphs. The results obtained for these two sets of COBOL programs agree with the result obtained by [8], namely, as the number of modules in a program increases, Halstead's E continuously overestimates the programming time. When a similar study was performed for smaller programs (e.g., programs containing only 1 to 4 procedure division paragraphs), it was observed that E value underestimates the programming time as shown in the table below.

Program-ID	EFFORT (E)	Estimated Time, T, in Hours	Reported Time in Hours
CIS 212-L2-TC1181	431250	6.7	11
CIS 212-L2-TC1183	361542	5.6	24
CIS 212-L2-TC1193	399532	6.2	26
CIS 212-L2-TC1194	546986	8.4	17
CIS 212-L2-TC1195	436812	6.7	15
CIS 212-L2-TC1199	414357	6.4	15
CIS 212-L2-TC1200	392176	6.1	13
CIS 212-L2-TC1208	413988	6.4	28.5
CIS 212-L2-TC1210	513006	7.9	27
CIS 212-L2-TC1211	511548	7.9	15.5

The programs in this table were collected from an undergraduate course on

"Computer Data Processing" at Ohio State University. The program deals primarily with the formatted listing of an input file, along with some control totals, e.g., total number of records in the file. Each student, while writing the program, is required to keep track of the development history using a shot log. The shot log provides the actual time spent by the programmer at various stages (e.g., in designing, in coding, in debugging, etc.).

Based on the behavior of E, as observed for COBOL programs of various sizes, it is evident that a new approach to measuring development effort is warranted. This chapter is devoted to addressing this question.

#### 4.2 Formulation of the Approach

Software typically consists of a set of modules. The total effort required to develop a piece of software can then be defined as the sum of the unit efforts of all the modules plus the effort needed to integrate these modules into a single system. In other words, the total development effort can be expressed as:

$$E_{\text{Total}} = \sum_{i=1}^q U(i) + E_I \quad (4.2)$$

and the estimated development time might be computed according (a la Software Science) to the expression

$$\text{Test} = E_{\text{Total}} / (18 \times 3600) \text{ hours}$$

where

$U(i)$  = Effort needed to write the  $i$ th module (for COBOL, this will mean the appropriate paragraph of procedure division together with its accompanying data division entries) [UNIT EFFORT of the  $i$ th module];

$E_I$  = Effort required to integrate all the modules to form a complete system (INTEGRATION EFFORT);

$q$  = Number of modules (paragraphs) in the procedure division.

In the present analyses, the unit effort  $U(i)$  of the  $i$ th module is considered to be the Halstead E measure for this module, together with its accompanying data division entries. The  $E_I$  of the system has been approached using the following three strategies.

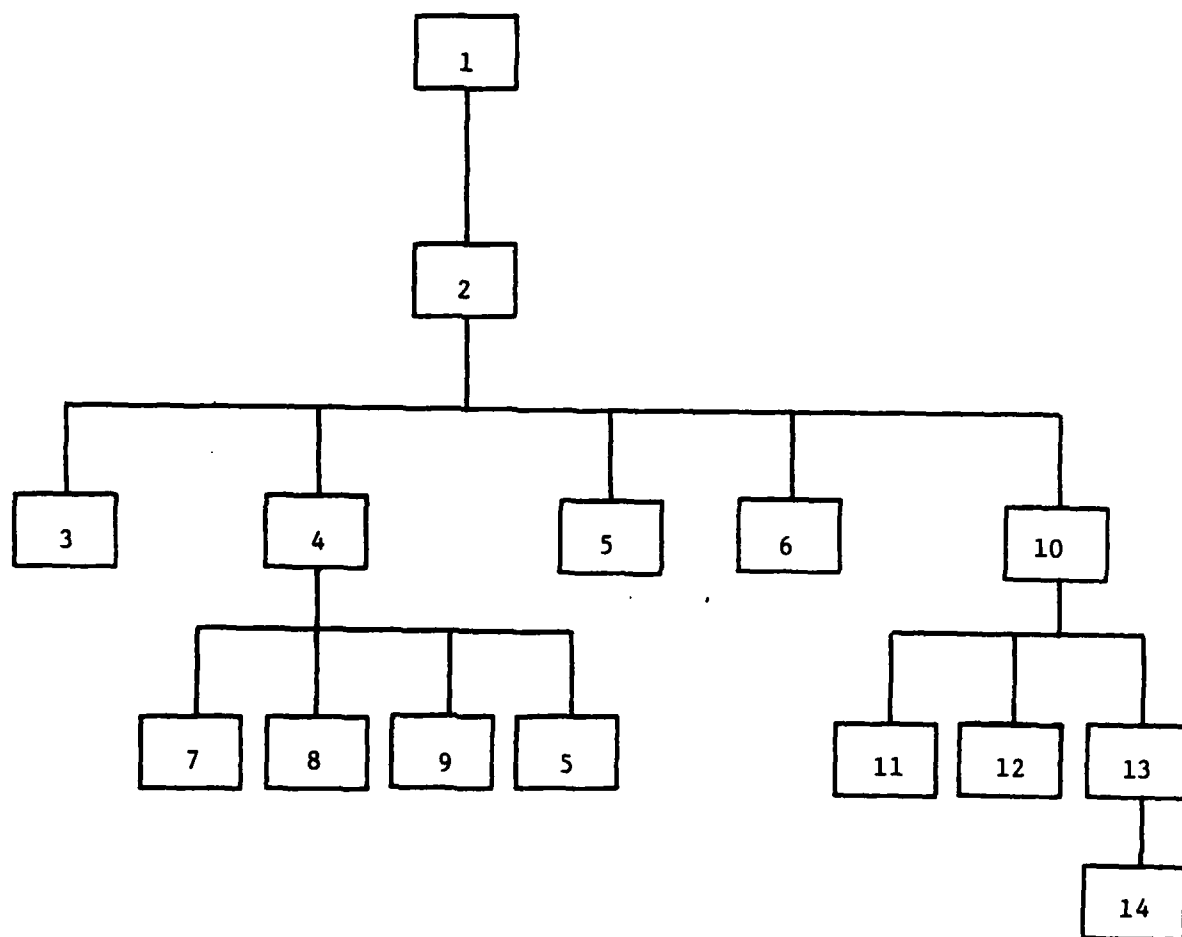
#### 4.2.1 Strategy 1

The integration effort for a subsystem is the effort required to integrate or combine all the modules contained in the subsystem. So, intuitively, only a fraction of the total efforts of all the modules contained in the subsystem would be contributing to the integration effort. Therefore, one possible approach to find the integration effort can be formalized as follows.

$$E_I \text{ of a subsystem} = K \left[ \begin{array}{l} \text{Effort of the union of all the} \\ \text{modules in the subsystem except} \\ \text{the driver (calling module)} \end{array} \right]$$

where  $K$  is a multiplication factor denoting a fraction of the union above. Let us arbitrarily choose this fraction to be  $1/2$ . The integration effort for each subsystem can be calculated using this strategy, which can then be extended to find the integration effort for the entire system. Once the  $E_I$  for the whole system is obtained, the use of equation (4.1) will allow  $E_{Total}$  and hence  $T_{est}$  for the software system to be calculated.

To illustrate, the structure chart for the program CIS 313-L2-TC0650 and the entire derivation using this strategy follows:



$$\begin{aligned}
 E(\text{sub } 13) &= E(\text{sub}(13, 14)) \\
 &= U(13) + U(14) + K E(14) \\
 &= U(13) + U(14) + K U(14), \quad \forall E(14) = U(14)
 \end{aligned}$$

$$\begin{aligned}
 E(\text{sub } 10) &= E(\text{sub}(10, 11, 12, 13, 14)) \\
 &= U(10) + U(11) + U(12) + E(\text{sub } 13) + \\
 &\quad K[U(11, 12, 13, 14)]
 \end{aligned}$$

$$E(\text{sub } 4) = E(\text{sub}(4, 7, 8, 9, 5))$$



$$\begin{aligned}
&= U(4) + U(5) + U(7) + U(8) + U(9) + \\
&\quad K[U(5, 7, 8, 9)] \\
E(\text{sub } 2) &= E(\text{sub}(2, 3, 4, 5, 6, 10)) \\
&= U(2) + U(3) + E(\text{sub } 4) + U(5) + U(6) + \\
&\quad E(\text{sub } 10) + K[U(3, 4, 5, \dots, 14)] \\
E_{\text{Total}} &= U(1) + E(\text{sub } 2) + K[U(2, 3, \dots, 14)] \\
&= \sum_{i=1}^{14} U(i) + K[U(14) + U(5, 7, 8, 9) + \\
&\quad U(11, 12, 13, 14) + U(3, 4, \dots, 14) + \\
&\quad U(2, 3, \dots, 14)] \tag{4.3}
\end{aligned}$$

Note: Notation  $U(n_1, n_2, \dots, n_i)$  means the effort of the union of the modules  $n_1, n_2, \dots, n_i$ .

#### 4.2.2 Strategy 2

By examining the second term of equation (4.3), it can be seen that  $E_I$  contains the effort value of some modules more than once. In particular, modules not directly called by the subsystem driver are continually contributing to the integration effort of the subsystem according to equation (4.3). The result of compounding the efforts for these modules may cause the estimated effort to be too large.

Therefore, when computing the integration effort, Strategy 2 will count only those modules which are directly called by the driver rather than all the modules below the calling module. That is,

$$E_I \text{ of a subsystem} = K[\text{Effort of the union of all} \\
\text{the modules directly called by} \\
\text{the subsystem driver}],$$

where  $K$  is still considered to be  $1/2$ .

The detailed derivation for calculating the  $E_{\text{Total}}$  for the same system using Strategy 2 follows.

$$\begin{aligned}
E(\text{sub } 13) &= E(\text{sub}(13, 14)) \\
&= U(13) + U(14) + K U(14)
\end{aligned}$$

$$\begin{aligned}
E(\text{sub } 10) &= E(\text{sub}(10, 11, 12, 13, 14)) \\
&= U(10) + U(11) + U(12) + E(\text{sub } 13) + \\
&\quad K[U(11, 12, 13)] \\
E(\text{sub } 4) &= E(\text{sub}(4, 7, 8, 9, 5)) \\
&= U(4) + U(5) + U(7) + U(8) + U(9) + \\
&\quad K[U(5, 7, 8, 9)] \\
E(\text{sub } 2) &= E(\text{sub}(2, 3, 4, 5, 6, 10)) \\
&= U(2) + U(3) + E(\text{sub } 4) + U(5) + U(6) + E(\text{sub } 10) + \\
&\quad K[U(3, 4, 5, 6, 10)] \\
E_{\text{Total}} &= U(1) + E(\text{sub } 2) + K U(2) \\
&\quad 14 \\
&= \sum_{i=1}^{14} U(i) + K[U(2) + U(14) + U(5, 7, 8, 9) + \\
&\quad U(11, 12, 13) + U(3, 4, 5, 6, 10)] \tag{4.4}
\end{aligned}$$

### Strategy 3

In the previous strategies for finding the integration effort of a subsystem, the multiplication factor  $K$  was arbitrarily chosen to be  $1/2$ . In practice, however, some modules are always more difficult to integrate than others, depending on such factors as the interaction between the calling module and the called module, the number of times a particular module is called from various parts of the system, etc. In other words, all the modules should not be weighted the same, and there should be some means of discriminating the multiplication factors associated with different subsystems. Hence, in order to provide better justification for the choice of  $K$ , Strategy 3 uses the following rule for selecting the multiplication factor to be associated with a subsystem.

Number of data elements which are common between the  
subsystem driver and the union of the modules under  
consideration

$$K = \frac{\text{Total number of data elements in the entire union of modules of the subsystem under consideration}}{\text{Total number of data elements in the entire union of modules of the subsystem under consideration}}$$

where  $0 \leq K \leq 1$ .

The exact value of  $K$  will depend on the amount of interaction among the modules contained in a subsystem. A subsystem which has a large number of variables in common between the subsystem driver and the rest of the modules

is evidently more difficult to integrate than the one having less number of data elements in common between the driver and the rest of the subsystem. In particular, if the calling module of the subsystem does not have any variable in common with the union of the rest of the modules, then  $K = 0$ . On the other hand, if the calling module has all its data elements in common with the union, then  $K = 1$ . In general, however, these extreme values of  $K$  do not seem to occur very often. The value of  $K$ , in most situations, is observed to be less than  $1/2$  and depends on the number and the size of the modules contained in the union under consideration. To find the integration effort of a subsystem using Strategy 3, the value of  $K$ , as calculated using the rule stated above, is incorporated into equation (4.3) of Strategy 1.

#### 4.3 Preliminary Results

All three strategies discussed in the previous section were applied to two sets of COBOL programs, as described earlier in the chapter (Section 4.1). The preliminary results obtained in all three cases are summarized in the table shown below. The detailed calculations for each program have been included in Appendix B.

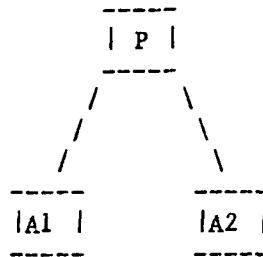
Set	Program ID	Estimated using Halstead's Eq.		Estimated using Strategy 1		Estimated using Strategy 2		Estimated using Strategy 3		Reported Time in Hours
		EFFORT	Time T in Hrs.	EFFORT	Time T1 in Hrs.	EFFORT	Time T2 in Hrs.	EFFORT	Time T3 in Hrs.	
1	CIS 313-L2 -TC0650	1104705	17	1475028	22.7	806751	12.4	977778	15.08	15.0
	CIS 313-L2 -TC0671	1241195	19	1399156	21.6	727586	11.2	1033443	15.9	10.5
	CIS 313-L2 -TC0645	1267142	19.5	1413010	21.8	775025	12	1121026	17.3	27.0
2	CIS 313-L3 -TC0671	5503529	85	6799232	105	2508999	38.7	2895952	44.7	40
	CIS 313-L3 -TC0645	6895476	106	6863622	105.9	3705425	57	5220200	80	31
	CIS 313-L3 -TC0622	6938627	107	7180899	110	2937307	45	4078366	63	49

Note that T1 (Test using Strategy 1) is much higher than the actual reported time, as expected from the nature of the equation (4.3). The summary of the results indicates that in some cases Strategy 2 works better than Strategy 3 and vice versa, although neither of these strategies work uniformly well in predicting actual time required to write the software. Both T2 and T3 appear to more uniformly approximate reported programming time than do T1 or T (the Halstead estimate), particularly on the larger program (Set 2). While this evidence is very preliminary and inconclusive, it does suggest that new approaches to measuring integration effort may yield more useful approximations of development time.

#### 4.4 Further Refinement

The results of the previous section suggest that further refinement of the models proposed is required to obtain more reasonable agreement between the estimated time and the reported time. Some of the issues which seem to be helpful for further refinements of the models are described below.

1. The interaction between the subsystem driver and each individual module contained in the subsystem should be more carefully considered. Consider an example as shown below, where the driver P calls two modules A1 and A2 sequentially.



Suppose the set of data elements that appear in P is A, B, C, M, N. Now two different cases can be considered depending on how the data elements appear in modules A1 and A2. For example, in one case A1 and A2 may contain the data elements A, B, C, L and A, B, C, H respectively. In the other case A1 and A2 may contain data elements A, B, L and C, H respectively. According to Strategy 3 (discussed in Section 4.2), in both the cases the multiplication factor  $K = 3/5$ . The integration effort for this subsystem is, therefore,

$$K[U(A1, A2)] \\ = 3/5[U(A1, A2)]$$

where  $U(A1, A2)$  denotes the effort when A1 and A2 are combined together.

It should be noted that the multiplication factor K is used to define the interaction between a calling module and the rest of the modules in the subsystem. In the present example, there is clearly more interaction between the driver and the called modules in the first case than there is in the second case. Intuitively, one might therefore expect the first case to require more integration effort than the second case. That is, the value of K in the first case should be greater than  $3/5$ , and in the second case it should be less than  $3/5$ . To overcome this problem, we might separately calculate K (using the rule of Strategy 3) between the driver and each individual module contained in the subsystem, and then take the average of all K's. This average value of K may be used as the

multiplication factor associated with the subsystem. In the first case of the current example, the interaction between P and A1 would be given by

$$K_1 = 3/4$$

and that between P and A2 is also

$$K_2 = 3/4.$$

Hence, K for the subsystem is found to be

$$k = \frac{K_1 + K_2}{2} = 3/4 \text{ (which is } > 3/5 \text{)}.$$

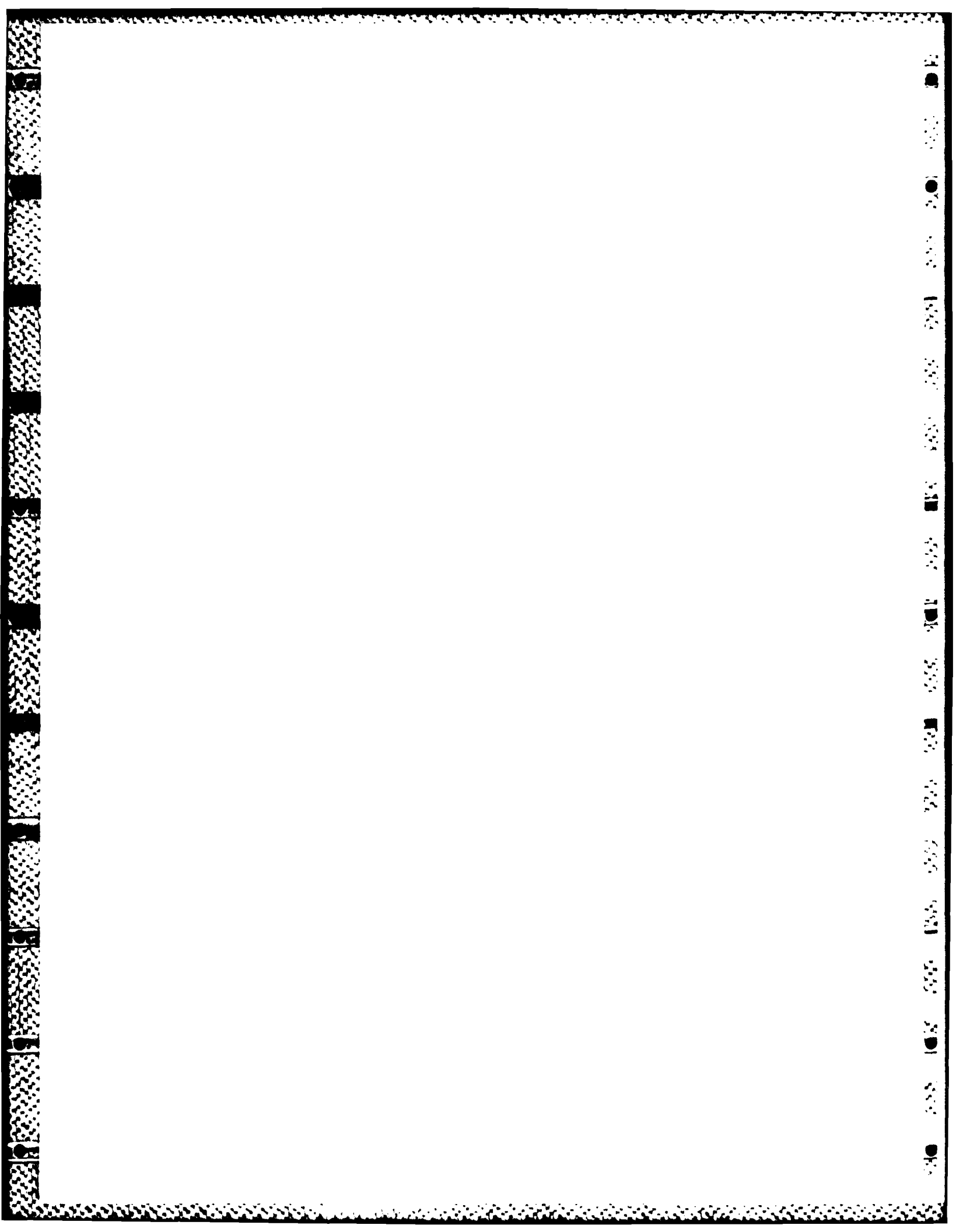
The similar calculations for the second case give

$$K_1 = 2/3, K_2 = 1/2 \text{ and } K = 7/12 \text{ (which is less than } 3/5 \text{)}.$$

The above discussion and the results of the previous section indicate that a reasonable model of integration effort might use Strategy 2 (discussed in Section 4.2) with the K value to be calculated using the strategy described above instead of the arbitrarily chosen value of  $K = 1/2$ .

2. In Chapter 2, it was realized that both the information flow complexity and the chunk model complexity give strong emphasis to the connectivity among system components. Therefore, it may be possible to apply the idea of information flow and chunk model for evaluating the integration effort of a system, since the integration effort primarily accounts for the connectivity among different system components.

It is our opinion that, once a more accurate method for calculating the integration effort associated with a system is found, more reliable estimates of the actual development time for a piece of software can be obtained.



## 5. Conclusion

The existence of the Software Science Analyzer developed at Ohio State University helps to collect and analyze a large number of COBOL programs. Various kinds of analyses using these programs are possible, including those initiated in this report. The major part of the report was devoted to the validation of the software science metrics on the basis of the analyses of a large number of COBOL programs. It was observed that for CIS 212 programs (small programs), the best estimate of the program length is attained for the entire program rather than for the data or procedure division alone. However, for the larger CIS 313 and University Systems programs, the length equation works equally well both for the entire program as well as for the procedure division alone. But since the data division is a significant part of any COBOL program and may require a considerable amount of programming effort, we recommend that it be included in software science studies. Secondly, software science postulates that the language level ( $\lambda$ ) would be constant for all programs written in a given language. The present analyses, however, indicate that the language level is not constant. Its use in other software science relationships is therefore suspect, and it is not recommended as a useful metric to be applied to an individual program. Finally, the estimated programming time (as calculated from the Effort metric) provided tantalizingly good values for many of the smaller student programs, but failed to produce good results when applied to larger programs. We feel that this is due to a faulty capturing of integration effort by the Halstead E measure. For small programs, integration is of minimal importance, so E may work well. For large programs, however, integration effort is critical.

We also studied the interrelationships among several software metrics, namely, Halstead's effort, McCabe's cyclomatic complexity, Kafura's information flow complexity and Davis' chunk model complexity. In particular, we studied the behavior of each of these measures as estimates of programming



time. We were unimpressed with the way each of the metrics behaved in this regard.

It is our feeling that we still do not have very useful measures of software effort, measures in which we can place much confidence. We have suggested some approaches for further study, and reasons why we believe they may be fruitful directions to pursue. The approaches are in some ways derivatives of existing measures, so that past work may not all be in vain. They attempt to promote the strengths of existing measures while correcting observed weaknesses. Without a fairly solid theory on which to rest the development of software metrics, this appears to be the best we can do. Laboratory experiments of the future will attest to the merits of any new approaches. The current metrics of software complexity, however, appear weak and very incomplete.

## References

- [1] Davis, John S.  
Chunks: A Basis for Complexity Measurements.  
Proceedings of the Symposium on Empirical Foundations of Information and Software Science, Nov., 1982.
- [2] Fung, K.C., Debnath, N.C., and Zweben, S.H.  
A Software Science Analyzer for COBOL.  
Technical Report OSU-CISRC-TR-83-2, Computer and Information Science Research Center, Ohio State University, Columbus, OH, Feb., 1983.
- [3] Halstead, M.  
Elements of Software Science.  
Elsevier North-Holland, New York, 1977.
- [4] Henry, S. and Kafura, D.  
Software Structure Metrics Based on Information Flow.  
IEEE Transactions on Software Engineering SE-7(5), Sep., 1981.
- [5] McCabe, T. J.  
A Complexity Measure.  
IEEE Transactions on Software Engineering SE-2, Dec., 1976.
- [6] Shen, V.Y. and Dunsmore, H.E.  
A Software Science Analysis of COBOL Programs.  
Technical Report CSD-TR-348, Department of Computer Science, Purdue University, West Lafayette, IN, Aug., 1980.
- [7] Smith, C.P.  
A Software Science Analysis of IBM Programming Products.  
IBM Santa Teresa Laboratory, TR03.081, Jan. 1980.
- [8] Woodfield, S.N.  
Enhanced Effort Estimation by Extending Basic Programming Models to Include Modularity Factors.  
PhD thesis, Department of Computer Science, Purdue University, West Lafayette, IN., Dec., 1980.
- [9] Woodfield, S.N., Dunsmore, H.E., and Shen, V.Y.  
The Effect of Modularization and Comments on Program Comprehension.  
Proc. Fifth International Conference on Software Engineering, 215-223, March, 1981.
- [10] Woodfield, S.N., Shen, V.Y., and Dunsmore, H.E.  
A Study of Several Metrics for Programming Effort.  
Journal Systems and Software, Dec., 1981.
- [11] Zweben, S.H. and Fung, K.C.  
Exploring Software Science Relations in COBOL and APL.  
Proc. COMPSAC 79, Nov., 1979.

## I. A

The detailed counting strategy used to calculate the fan-in and fan-out associated with each module of a COBOL program is described. The explicit calculations for finding the information flow complexity metric and the chunk model complexity metric corresponding to each program, considered in Chapter 3, are also included.

### I.1 Counting Strategy Used to Find the Information Flow Complexity for COBOL Programs

The information flow complexity of a COBOL program was determined based on the following strategies.

1. Each paragraph of the procedure division (PD) was considered as a separate procedure.
2. Two different complexity values were derived using two different length values as follows.

#### a. Length

= Number of statements (NOS) of the particular procedure under consideration

= Number of verbs in the paragraph

#### b. Length

= (NOS of the particular paragraph in PD) + (NOS in the data division entries associated with this paragraph)

= (#verbs in the PD paragraph) + (# periods in the associated data division entries)

The Data Structures (DS) retrieved and updated are counted by assessing the number of referenced and assigned data items, respectively, based on the semantics of the various COBOL statements. Some of the rules followed are listed below.



COBOL Statements

- (a) MOVE identifier-1 TO identifier-2.  
 MOVE {constant  
       literal} TO identifier.  
 e.g., MOVE 1 TO FLAG;  
       MOVE 'BILL' TO NAME-IN.
- (b) ADD constant to identifier  
 e.g., ADD 2 TO COUNT.  
  
 ADD id-1, id-2 GIVING id-3  
 e.g., ADD A, B GIVING C.  
  
 ADD id-1 TO id-2  
 e.g., ADD A TO B.
- (c) IF A = B THEN .....  
       IF A IS NUMERIC THEN .....
- (d) READ filename AT END  
       MOVE 'YES' TO FLAG.
- (e) WRITE OUT-REC FROM  
       detail-line AFTER  
       ADVANCING 2 LINES.  
 e.g.,  
       { WRITE OUT-REC FROM Heading-1.  
       WRITE OUT-REC FROM Heading-2.
- (f) SORT Sort-file .....  
       USING file-1  
       GIVING file-2.

Rules

DS retrieved = 1, DS updated = 1  
 DS retrieved = 0, DS updated = 1

DS retrieved = 1, DS updated = 1

DS retrieved = 2, DS updated = 1

DS retrieved = 2, DS updated = 1

DS retrieved = 2, DS updated = 0  
 DS retrieved = 1, DS updated = 0

DS retrieved = 1, DS updated = 1

DS retrieved = 1, DS updated = 1

DS retrieved = 2, DS updated = 1

DS retrieved = 1, DS updated = 1

### **I.2 Calculations for Finding Information Flow Complexity for Each Program**

The explicit values of fan-in and fan-out corresponding to each module (paragraph) along with their information flow complexity values are shown below. For each module two different values of length are considered, hence two different complexity values are shown. The first value corresponds to the length of the procedure division paragraph together with the associated data division entries. The second value of the length refers to the length of the procedure division paragraph only. The fan-in and fan-out for each module consists of two distinct operands. The first operand denotes the number of local flow(s) to (or from) the module considered. The second operand refers to the number of data structures from which the module retrieves information (or which the module updates).

### **I.3 Calculations for Finding Chunk Model Complexity for Each Program**

In order to find the chunk model complexity, the fan-in for each chunk (performed paragraph) is determined. The fan-in for each chunk is the sum of the number of control connections and the number of data connections to that chunk (as described in Section 3.4). The complexity of the entire program can then be calculated using the formula shown in Section 3.4. In this section, the fan-in value for each chunk is explicitly shown as the sum of two distinct operands. The first operand refers to the number of control connections to the particular chunk, and the second operand denotes the number of data connections to that chunk. The specific chunks having control or data connections to each chunk are indicated to the right of each fan-in computation (e.g., 2 => d1 means that chunk 2 has a data connection to chunk 1). The detailed calculations for finding the chunk model complexity for each program are shown below. The fan-in for the  $i$ th chunk is denoted by  $f_i$ .

INFORMATION FLOW COMPLEXITY

TC0671, Lab 2, CIS 313

MODULE	FAN-IN	FAN-OUT	LENGTH (NOS)		CFC = Length * (Fan-in * Fan-out) <sup>2</sup>	
1	1+4=5	1+4=5	41	8	25625	5000
2	2+10=12	5+22=27	76	24	7978176	2519424
3	1+1=2	4+2=6	33	11	4752	1584
4	2+1=3	0+2=2	24	2	864	72
5	1+3=4	0+4=4	31	6	7936	1536
6	1+2=3	0+3=3	27	4	2187	324
7	1+2=3	0+2=2	24	2	864	72
8	1+2=3	0+4=4	30	6	4320	288
9	4+4=8	3+4=7	36	9	112896	28224
10	2+14=16	1+3=4	85	56	348160	229375
11	1+8=9	0+4=4	42	8	54432	10368
12	1+8=9	0+4=4	42	8	54432	10368
13	2+8=10	0+4=4	42	8	67200	12800
					8661844	2819435

INFORMATION FLOW COMPLEXITY

TC0650

MODULE	FAN-IN	FAN-OUT	LENGTH (NOS)		CFC = Length* (Fan-in * Fan-out) <sup>2</sup>	
			D+P	P		
1	1+2=3	1+2=3	29	6	2349	486
2	1+2=3	5+12=17	38	12	98838	31212
3	1+15=16	0+11=11	69	19	2137344	588544
4	1+1=2	4+2=6	30	12	4320	1728
5	2+2=4	0+4=4	45	8	11520	2048
6	1+8=9	0+9=9	60	19	393660	124659
7	1+3=4	0+4=4	43	6	11008	1536
8	1+3=4	0+4=4	45	8	11520	2048
9	1+3=4	0+3=3	44	7	6336	1008
10	1+1=2	3+0=3	7	3	252	108
11	1+13=14	0+9=9	50	17	793800	269892
12	1+14=15	0+9=9	86	19	1567350	346275
13	1+9=10	1+1=2	33	18	13200	7200
14	1+2=3	0+2=2	12	2	432	72
					5051929	1376816

INFORMATION FLOW COMPLEXITY

TC0645

MODULE	FAN-IN	FAN-OUT	LENGTH (NOS)		CFC = Length* (Fan-in * Fan-out) <sup>2</sup>	
			D+P	P		
1	1+4=5	1+4=5	38	8	23750	5000
2	2+8=10	5+17=22	75	27	3630000	1306800
3	1+1=2	3+2=5	30	11	3000	1100
4	1+2=3	0+4=4	41	6	5904	864
5	1+8=9	0+8=8	65	22	336960	114048
6	1+3=4	0+4=4	41	6	10496	1536
7	1+3=4	0+4=4	41	6	10496	1536
8	1+3=4	0+5=5	55	19	22000	7600
9	4+5=9	3+4=7	40	13	158760	51597
10	2+11=13	0+3=3	58	39	88218	59319
11	1+5=6	0+5=5	42	8	37800	7200
12	1+5=6	0+5=5	42	8	37800	7200
13	1+5=6	0+5=5	42	8	37800	7200
					4402984	1571000



## INFORMATION FLOW COMPLEXITY

TC0645, Lab 3, CIS 313

MODULE	FAN-IN (I)	FAN-OUT (O)	Length (L)		IFC = $L \cdot (I+O)^2$	
1	4+8=12	7+11=18	122	25	5692032	1166400
2	3+8=11	5+18=23	75	28	4800675	1792252
3	1+1=2	3+2=5	30	11	30000	1100
4	1+2=3	0+4=4	41	6	5904	864
5	1+8=9	0+8=8	65	22	336960	114048
6	1+3=4	0+4=4	41	6	10496	1536
7	1+3=4	0+4=4	41	6	10496	1536
8	1+3=4	0+5=5	55	19	22000	7600
9	6+9=15	3+10=13	68	19	2585700	722475
10	1+11=12	0+3=3	55	36	71280	46656
11	1+5=6	0+5=5	42	8	37800	7200
12	1+5=6	0+5=5	42	8	37800	7200
13	1+5=6	0+5=5	42	8	37800	7200
14	2+5=7	0+5=5	42	8	51450	9800
15	3+2=5	0+2=2	17	2	1700	200
16	1+2=3	0+2=2	20	2	720	72
17	6+2=8	0+1=1	51	4	3264	256
18	5+2=7	4+0=4	35	4	27440	3136
19	1+3=4	1+2=3	41	6	5904	864
20	4+7=11	3+9=12	83	15	1446192	261360
21	3+9=12	0+14=14	80	24	2257920	677376
22	1+1=2	6+0=6	32	17	4608	2448
23	2+10=12	1+15=16	110	20	4055040	737280
24	2+5=7	2+6=8	73	16	228928	50176
25	2+7=9	2+7=9	85	14	557685	91854
26	2+4=6	2+5=7	83	12	146412	21168
27	2+5=7	2+6=8	73	16	228928	50176
28	2+4=6	2+6=8	74	16	170496	36864
29	6+5=11	0+4=4	62	5	120032	9680
30	6+6=12	0+14=14	68	7	1919232	197568
31	5+6=11	1+8=9	80	9	784080	88209
32	1+4=5	0+5=5	33	7	20625	4375
					25709599	6118929

INFORMATION FLOW COMPLEXITY

TC0671, Lab 3, CIS 313

MODULE	FAN-IN (I)	FAN-OUT (O)	Length (L)		IFC = $L \cdot (I \cdot O)^2$	
1	1+1=2	3+3=6	150	13	21600	1872
2	2+4=6	1+4=5	27	5	24300	4500
3	3+2=5	4+0=4	22	4	8800	1600
4	2+2=4	0+2=2	21	2	1344	128
5	2+2=4	0+2=2	20	2	1280	128
6	4+2=6	0+1=1	56	4	2016	144
7	4+2=6	5+0=5	38	5	34200	4500
8	1+3=4	1+2=3	47	6	6768	864
9	2+2=4	3+3=6	42	8	24192	4608
10	1+7=8	0+4=4	37	8	37888	8192
11	2+12=14	0+15=15	70	14	3087000	617400
12	1+1=2	6+0=6	35	17	5040	2448
13	4+5=9	2+7=9	58	12	380538	78732
14	6+10=16	0+14=14	73	14	3662848	702464
15	1+0=1	0+1=1	19	1	19	1
16	1+5=6	2+11=13	85	14	517140	85176
17	1+3=4	2+4=6	65	14	37440	8064
18	1+4=5	2+5=7	82	12	100450	14700
19	1+4=5	2+5=7	80	10	98000	12250
20	1+3=4	2+4=6	67	14	38592	8064
21	1+2=3	2+3=5	65	13	14625	2925
22	7+11=18	1+9=10	45	16	1458000	518400
23	3+14=17	1+14=15	57	16	3706425	1040400
24	1+5=6	0+4=4	41	7	23616	4032
25	2+11=13	5+22=27	73	24	8993673	2956824
26	1+1=2	4+2=6	33	11	4752	1584
27	2+1=3	0+2=2	24	2	864	72
28	1+3=4	0+4=4	31	6	7936	1536
29	1+2=3	0+3=3	27	4	2187	324
30	1+2=3	0+2=2	24	2	864	72
31	1+2=3	0+4=4	30	6	4320	864
32	7+5=12	3+5=8	53	10	488448	92160
33	2+15=17	1+3=4	85	56	393040	258944
34	1+7=8	0+4=4	42	8	43008	8192
35	1+7=8	0+4=4	42	8	43008	8192
36	3+7=10	0+4=4	43	8	68800	12800
					23343021	5913156

INFORMATION FLOW COMPLEXITY

TC0622, Lab 3, CIS 313

MODULE	FAN-IN (I)	FAN-OUT (O)	Length (L)		IFC = $L \cdot (I+O)^2$	
1	4+8=12	6+12=18	124	30	5785344	1399680
2	1+1=2	0+2=2	12	2	192	32
3	1+5=6	1+3=4	34	10	19584	5760
4	2+11=13	5+18=23	63	29	5632263	2592629
5	1+7=8	4+2=6	31	12	71424	27648
6	1+2=3	0+2=2	16	3	576	108
7	1+2=3	0+2=2	16	3	576	108
8	1+2=3	0+2=2	17	4	612	144
9	2+1=3	0+1=1	12	2	108	18
10	1+3=4	0+3=3	21	6	3024	864
11	4+15=19	3+13=16	82	29	7578112	2680064
12	1+5=6	0+5=5	35	11	31500	9900
13	2+7=9	0+5=5	37	12	74925	24300
14	2+11=13	0+3=3	59	39	89739	59319
15	2+8=10	0+6=6	31	9	111600	32400
16	2+9=11	0+7=7	42	10	249018	59290
17	4+2=6	0+1=1	45	2	1620	72
18	5+2=7	4+0=4	31	4	24304	3136
19	2+8=10	1+7=8	59	11	377600	70400
20	4+5=9	4+7=11	64	13	627264	127413
21	3+7=10	2+9=11	85	16	1028500	193600
22	8+6=14	0+5=5	39	11	191100	53900
23	1+1=2	6+0=6	18	6	2592	864
24	2+11=13	0+12=12	37	12	900432	292032
25	2+8=10	4+14=18	108	25	3499200	810000
26	2+6=8	4+6=10	80	28	512000	179200
27	2+4=6	5+4=9	54	17	157464	49572
28	2+6=8	4+7=11	83	18	642752	139392
29	2+6=8	4+6=10	80	28	512000	179200
30	2+5=7	4+5=9	80	28	317520	111132
31	6+8=14	0+18=18	75	18	4762800	1143072
32	1+3=4	0+3=3	31	3	4464	432
33	1+15=16	1+14=15	60	17	3456000	979200
34	1+5=6	0+5=5	36	11	32400	9900
35	7+3=10	1+3=4	29	5	46400	8000
36	7+3=10	0+2=2	15	3	6000	1200
					36751009	11243981

Program: CIS 313 - L2 - TC0650

$f_1 = 0+1 = 1$	$2 \rightarrow d1$
$f_2 = 1+1 = 2$	$1 \rightarrow c2, 13 \rightarrow d2$
$f_3 = 1+0 = 1$	$2 \rightarrow c3$
$f_4 = 1+1 = 2$	$2 \rightarrow c4, 13 \rightarrow d4$
$f_5 = 2+1 = 3$	$2 \rightarrow c5, 4 \rightarrow c5, 13 \rightarrow d5$
$f_6 = 1+2 = 3$	$2 \rightarrow c6, 10 \rightarrow d6, 13 \rightarrow d6$
$f_7 = 1+1 = 2$	$4 \rightarrow c7, 13 \rightarrow d7$
$f_8 = 1+1 = 2$	$4 \rightarrow c8, 13 \rightarrow d8$
$f_9 = 1+1 = 2$	$4 \rightarrow c9, 13 \rightarrow d9$
$f_{10} = 1+0 = 1$	$2 \rightarrow c10$
$f_{11} = 1+0 = 1$	$10 \rightarrow c11$
$f_{12} = 1+1 = 2$	$10 \rightarrow c12, 14 \rightarrow d12$
$f_{13} = 1+1 = 2$	$10 \rightarrow c13, 14 \rightarrow d13$
$f_{14} = 1+0 = 1$	$13 \rightarrow c14.$

$$C = (c_1+c_3+c_{10}+c_{11}+c_{14}) \cdot \sum_{m=0}^1 R^m + (c_2+c_4+c_7+c_8+c_9+c_{12}+c_{13}) \cdot \sum_{m=0}^2 R^m + (c_5+c_6) \cdot \sum_{m=0}^3 R^m.$$

$$\begin{aligned} C_{P+D} &= (29+69+7+50+12) 1 \cdot 666 + (38+30+43+45+44+86+33) 2 \cdot 11 + \\ &\quad (45+60) 2 \cdot 406 \\ &= 1204 \end{aligned}$$

$$\begin{aligned} C_P &= (6+19+3+17+2) 1 \cdot 666 + (12+12+6+8+7+19+18) 2 \cdot 11 + (8+19) 2 \cdot 406. \\ &= 316 \end{aligned}$$

Note:  $C_{P+D}$  is the complexity of the program when length for each module contains the length of the PD paragraph along with the associated DD entries.  $C_P$  is the complexity of the program when length of each module refers to the length of the PD paragraph only.

Program: CIS 313 - L2 - TC0671

$f_1 = 0+1 = 1$	$2 \rightarrow d1$
$f_2 = 1+2 = 3$	$1 \rightarrow c2, 9 \rightarrow d2, 10 \rightarrow d2$
$f_3 = 1+1 = 2$	$2 \rightarrow c3, 10 \rightarrow d3$
$f_4 = 2+1 = 3$	$2 \rightarrow c4, 3 \rightarrow c4, 10 \rightarrow d4$
$f_5 = 1+2 = 3$	$2 \rightarrow c5, 9 \rightarrow d5, 10 \rightarrow d5$
$f_6 = 1+1 = 2$	$3 \rightarrow c6, 10 \rightarrow d6$
$f_7 = 1+1 = 2$	$3 \rightarrow c7, 10 \rightarrow d7$
$f_8 = 1+1 = 2$	$3 \rightarrow c8, 10 \rightarrow d8$
$f_9 = 1+1 = 2$	$2 \rightarrow c9, 10 \rightarrow d9$
$f_{10} = 1+0 = 1$	$9 \rightarrow c10$
$f_{11} = 1+0 = 1$	$2 \rightarrow c11 \text{ and } 2 \rightarrow d11$
$f_{12} = 1+0 = 1$	$9 \rightarrow c12 \text{ and } 9 \rightarrow d12$
$f_{13} = 2+0 = 2$	$9 \rightarrow c13, d13; 10 \rightarrow c13, d13$

$$C = (c_1+c_{10}+c_{11}+c_{12}) \cdot \sum_{m=0}^1 R^m + (c_3+c_6+c_7+c_8+c_9+c_{13}) \cdot \sum_{m=0}^2 R^m + (c_2+c_4+c_5) \cdot \sum_{m=0}^3 R^m.$$

$$\begin{aligned} C_{P+D} &= (41+85+42+42) 1 \cdot 666 + (33+27+24+30+36+42) 2 \cdot 11 + \\ &\quad (76+24+31) 2 \cdot 406 \\ &= 1070 \end{aligned}$$

$$\begin{aligned} C_P &= (8+56+8+8) 1 \cdot 666 + (11+4+2+6+9+8) 2 \cdot 11 + (24+2+6) 2 \cdot 406 \\ &= 295 \end{aligned}$$

Program: CIS 313 - L2 - TC0645

$f_1 = 0+1 = 1$	$2 \rightarrow d1$
$f_2 = 1+2 = 3$	$1 \rightarrow c2, 9 \rightarrow d2, 10 \rightarrow d2$
$f_3 = 1+1 = 2$	$2 \rightarrow c3, 10 \rightarrow d3$
$f_4 = 1+1 = 2$	$2 \rightarrow c4, 10 \rightarrow d4$
$f_5 = 1+1 = 2$	$2 \rightarrow c4, 10 \rightarrow d5$
$f_6 = 1+1 = 2$	$3 \rightarrow c6, 10 \rightarrow d6$
$f_7 = 1+1 = 2$	$3 \rightarrow c7, 10 \rightarrow d7$
$f_8 = 1+1 = 2$	$3 \rightarrow c8, 10 \rightarrow d8$
$f_9 = 1+1 = 2$	$2 \rightarrow c9, 10 \rightarrow d9$
$f_{10} = 1+0 = 1$	$9 \rightarrow c10$
$f_{11} = 1+0 = 1$	$2 \rightarrow c11, d11$
$f_{12} = 1+0 = 1$	$9 \rightarrow c12, d12$
$f_{13} = 1+1 = 2$	$9 \rightarrow c13, 10 \rightarrow d13$

$$C = (c_1 + c_{10} + c_{11} + c_{12}) \cdot \sum_{m=0}^1 R^m + (c_3 + c_4 + c_5 + c_6 + c_7 + c_8 + c_9 + c_{13}) \cdot \sum_{m=0}^2 R^m + (c_2) \cdot \sum_{m=0}^3 R^m.$$

$$C_{D+P} = (38+58+42+42) 1 \cdot 666 + (30+41+65+41+41+55+40+42) 2 \cdot 11 + (75 \times 2 \cdot 406) \\ = 1229$$

$$C_P = (8+39+8+8) 1 \cdot 666 + (11+6+22+6+6+19+13+8) 2 \cdot 11 + (27 \times 2 \cdot 406) \\ = 362$$

Program: CIS 313 - L3 - TC0645

$$\begin{array}{ll}
 f_1 = 0+17 = 17 & f_{17} = 2+5 = 7 \\
 f_2 = 1+4 = 5 & f_{18} = 1+0 = 1 \\
 f_3 = 1+2 = 3 & f_{19} = 1+9 = 10 \\
 f_4 = 1+2 = 3 & f_{20} = 1+10 = 11 \\
 f_5 = 1+2 = 3 & f_{21} = 1+1 = 2 \\
 f_6 = 1+2 = 3 & f_{22} = 1+0 = 1 \\
 f_7 = 1+2 = 3 & f_{23} = 1+8 = 9 \\
 f_8 = 1+2 = 3 & f_{24} = 1+7 = 8 \\
 f_9 = 1+3 = 4 & f_{25} = 1+6 = 7 \\
 f_{10} = 1+2 = 3 & f_{26} = 1+5 = 6 \\
 f_{11} = 1+1 = 2 & f_{27} = 1+4 = 5 \\
 f_{12} = 1+1 = 2 & f_{28} = 1+2 = 3 \\
 f_{13} = 1+4 = 5 & f_{29} = 5+9 = 14 \\
 f_{14} = 2+11 = 13 & f_{30} = 6+9 = 15 \\
 f_{15} = 3+4 = 7 & f_{31} = 1+0 = 1 \\
 f_{16} = 1+2 = 3 & f_{32} = 1+0 = 1.
 \end{array}$$

Use of the notation

$$R_1 = \sum_{m=0}^1 R^m$$

gives

$$\begin{aligned}
 C = & (c_{18}+c_{22}+c_{31}+c_{32}) R_1 + (c_{11}+c_{12}+c_{21}) R_2 + (c_3+c_4+c_5+c_6+c_7+c_8+c_{10}+c_{16}+c_{28}) R_3 + \\
 & c_9 R_4 + (c_2+c_{13}+c_{27}) R_5 + c_{26} R_6 + (c_{15}+c_{17}+c_{25}) R_7 + c_{24} R_8 + c_{23} R_9 + \\
 & (c_{19} R_{10}) + c_{20} R_{11} + c_{14} R_{13} + c_{29} R_{14} + (c_{30} R_{15}) + (c_1 R_{17})
 \end{aligned}$$

$$\begin{aligned}
 C_{P+D} = & (25+32+80+33) 1 \cdot 666 + (42+42+80) 2 \cdot 11 + (30+41+65+41+41+55+55+20+74) 2 \cdot 406 + \\
 & (68 \times 2 \cdot 603) + (75+42+73) 2 \cdot 734 + (83 \times 2 \cdot 822) + (17+51+85) 2 \cdot 88 + (73 \times 2 \cdot 919) + \\
 & (110 \times 2 \cdot 945) + (41 \times 2 \cdot 962) + (83 \times 2 \cdot 973) + (42 \times 2 \cdot 978) + (62 \times 2 \cdot 981) + \\
 & (68 \times 2 \cdot 983) + (122 \times 2 \cdot 984) \\
 = & 4814
 \end{aligned}$$

$$\begin{aligned}
 C_P = & (4+17+9+7) 1 \cdot 666 + (8+8+24) 2 \cdot 11 + (11+6+22+6+6+19+36+2+16) 2 \cdot 406 + \\
 & (19 \times 2 \cdot 603) + (28+8+16) 2 \cdot 734 + (12 \times 2 \cdot 822) + (2+4+14) 2 \cdot 88 + (16 \times 2 \cdot 919) + \\
 & (20 \times 2 \cdot 945) + (6 \times 2 \cdot 962) + (15 \times 2 \cdot 973) + (8 \times 2 \cdot 978) + (5 \times 2 \cdot 981) + \\
 & (7 \times 2 \cdot 983) + (25 \times 2 \cdot 984) \\
 = & 1031
 \end{aligned}$$

Program: CIS 313 - L3 - TC0671

$$\begin{array}{ll}
 f_1 = 0+2 = 2 & f_{19} = 1+2 = 3 \\
 f_2 = 1+7 = 8 & f_{20} = 1+3 = 4 \\
 f_3 = 1+0 = 1 & f_{21} = 1+1 = 2 \\
 f_4 = 2+5 = 7 & f_{22} = 6+0 = 6 \\
 f_5 = 2+2 = 4 & f_{23} = 1+0 = 1 \\
 f_6 = 2+4 = 6 & f_{24} = 1+0 = 1 \\
 f_7 = 1+0 = 1 & f_{25} = 1+4 = 5 \\
 f_8 = 1+8 = 9 & f_{26} = 1+2 = 3 \\
 f_9 = 1+1 = 2 & f_{27} = 2+2 = 4 \\
 f_{10} = 1+3 = 4 & f_{28} = 1+2 = 3 \\
 f_{11} = 2+2 = 4 & f_{29} = 1+2 = 3 \\
 f_{12} = 1+0 = 1 & f_{30} = 1+2 = 3 \\
 f_{13} = 1+0 = 1 & f_{31} = 1+2 = 3 \\
 f_{14} = 6+2 = 8 & f_{32} = 1+3 = 4 \\
 f_{15} = 1+0 = 1 & f_{33} = 1+0 = 1 \\
 f_{16} = 1+10 = 11 & f_{34} = 1+1 = 2 \\
 f_{17} = 1+4 = 5 & f_{35} = 1+1 = 2 \\
 f_{18} = 1+2 = 3 & f_{36} = 3+3 = 6
 \end{array}$$

$$\begin{aligned}
 C = & (c_3+c_7+c_{12}+c_{13}+c_{15}+c_{23}+c_{24}+c_{33}) R_1 + (c_1+c_9+c_{21}+c_{34}+c_{35}) R_2 + \\
 & (c_{18}+c_{19}+c_{26}+c_{28}+c_{29}+c_{30}+c_{31}) R_3 + (c_5+c_{10}+c_{11}+c_{20}+c_{27}+c_{32}) R_4 + \\
 & (c_{17}+c_{25}) R_5 + (c_6+c_{22}+c_{36}) R_6 + (c_4 \cdot R_7) + (c_2+c_{14}) R_3 + (c_8 \cdot R_9) + \\
 & (c_{16} \cdot R_{11})
 \end{aligned}$$

$$\begin{aligned}
 C_{P+D} = & (22+38+35+58+19+57+41+85) 1 \cdot 666 + (150+42+65+42+42) 2 \cdot 11 + \\
 & (82+80+33+31+27+24+30) 2 \cdot 406 + (20+37+70+67+24+53) 2 \cdot 603 + \\
 & (65+73) 2 \cdot 734 + (56+45+43) 2 \cdot 822 + (21 \times 2 \cdot 88) + (27+73) 2 \cdot 919 + \\
 & (47 \times 2 \cdot 945) + (85 \times 2 \cdot 973) \\
 = & 4280
 \end{aligned}$$

$$\begin{aligned}
 C_P = & (4+5+17+12+1+16+7+56) 1 \cdot 666 + (13+8+13+8+8) 2 \cdot 11 + (12+10+11+6+4+2+6) 2 \cdot 406 + \\
 & (2+8+14+14+2+10+) 2 \cdot 603 + (14+24) 2 \cdot 734 + (4+16+8) 2 \cdot 822 + (2 \times 2 \cdot 88) + \\
 & (5+14) 2 \cdot 919 + (6 \times 2 \cdot 945) + (14 \times 2 \cdot 973) \\
 = & 860
 \end{aligned}$$



Program: CIS 313 - L3 - TC0622

$f_1 = 0+6 = 6$	$f_{19} = 1+10 = 11$
$f_2 = 1+1 = 2$	$f_{20} = 1+8 = 9$
$f_3 = 1+2 = 3$	$f_{21} = 1+8 = 9$
$f_4 = 1+6 = 7$	$f_{22} = 8+8 = 16$
$f_5 = 1+2 = 3$	$f_{23} = 1+0 = 1$
$f_6 = 1+2 = 3$	$f_{24} = 2+0 = 2$
$f_7 = 1+2 = 3$	$f_{25} = 1+11 = 12$
$f_8 = 1+2 = 3$	$f_{26} = 1+8 = 9$
$f_9 = 2+1 = 3$	$f_{27} = 1+8 = 9$
$f_{10} = 1+2 = 3$	$f_{28} = 1+9 = 10$
$f_{11} = 1+4 = 5$	$f_{29} = 1+8 = 9$
$f_{12} = 1+1 = 2$	$f_{30} = 1+8 = 9$
$f_{13} = 2+3 = 5$	$f_{31} = 6+0 = 6$
$f_{14} = 1+3 = 4$	$f_{32} = 1+1 = 2$
$f_{15} = 2+7 = 9$	$f_{33} = 1+0 = 1$
$f_{16} = 2+2 = 4$	$f_{34} = 1+0 = 1$
$f_{17} = 2+3 = 5$	$f_{35} = 6+1 = 7$
$f_{18} = 1+0 = 1$	$f_{36} = 6+1 = 7$

$$C = (c_{18}+c_{23}+c_{33}+c_{34}) R_1 + (c_2+c_{12}+c_{24}+c_{32}) R_2 + (c_3+c_5+c_6+c_7+c_8+c_9+c_{10}) R_3 + \\ (c_{14}+c_{16}) R_4 + (c_{11}+c_{13}+c_{17}) R_5 + (c_1+c_{31}) R_6 + (c_4+c_{35}+c_{36}) R_7 + \\ (c_{15}+c_{20}+c_{21}+c_{26}+c_{27}+c_{29}+c_{30}) R_9 + (c_{28} \times R_{10}) + (c_{19} \times R_{11}) + (c_{25} \times R_{12}) + \\ (c_{22} \times R_{16})$$

$$C_{P+D} = (31+18+60+36) 1 \cdot 666 + (12+35+37+31) 2 \cdot 11 + (34+31+16+16+17+12+21) 2 \cdot 406 + \\ (59+42) 2 \cdot 603 + (82+37+45) 2 \cdot 734 + (124+75) 2 \cdot 822 + (63+29+15) 2 \cdot 88 + \\ (31+64+85+80+54+80+80) 2 \cdot 949 + (83 \times 2 \cdot 962) + (59 \times 2 \cdot 973) + (108 \times 2 \cdot 98) + \\ (39 \times 2 \cdot 992) \\ = 4678$$

$$C_p = (4+6+17+11) 1 \cdot 666 + (2+11+12+3) 2 \cdot 11 + (10+12+3+3+4+2+6) 2 \cdot 406 + \\ (39+10) 2 \cdot 603 + (29+12+2) 2 \cdot 734 + (30+18) 2 \cdot 822 + (29+5+3) 2 \cdot 88 + \\ (9+13+16+28+17+28+28+) 2 \cdot 945 + (18 \times 2 \cdot 962) + (11 \times 2 \cdot 973) + (25 \times 2 \cdot 98) + (11 \times 2 \cdot 992) \\ = 1309$$

## II. B

Detailed Computations Applied to the Models of Chapter 4

The explicit expressions for E total, calculated using all three strategies described in Chapter 4, are presented in this section. The values of the unit effort for each module in a program are also shown. In order to determine the unit effort for each module (procedure division paragraph) together with its data division entries, each module was run through the Software Science Analyzer developed at Ohio State University [2].

PREVIOUS PAGE  
IS BLANK

## 1. Program: CIS 313-L2-TC0650

The structure chart for this program and the calculations for finding  $E_{\text{total}}$  were shown in Section 4.2. The unit effort for each module obtained from the analyzer report are shown below.

Module Number	Effort, E	Union of the Modules	Effort, E
1	18690	U(5,7,8,9)	69198
2	29471	U(11,12,13,14)	226655
3	64265	U(2,3, ... ,14)	912750
4	15232	U(3,4, ... ,14)	734895
5	27518	U(11,12,13)	218783
6	48370	U(3,4,5,6,10)	280491
7	28965		
8	30554		
9	32729		
10	20000		
11	33865		
12	95320		
13	35800		
14	18000		

By Equation (4.3), strategy 1 yields

$$\begin{aligned}
 E_{\text{total}} &= \sum_{i=1}^{14} U(i) + 1/2 [U(14) + U(5,7,8,9) + U(11,12,13,14) + U(3,4,\dots,14) \\
 &\quad + U(2,3,\dots,14)] \quad (1) \\
 &= 1475028
 \end{aligned}$$

$$T1 = \frac{1475028}{18 \times 3600} \text{ hrs.} = 22.7 \text{ hrs.}$$

By Equation (4.4), strategy 2 yields

$$E_{\text{total}} = \sum_{i=1}^{14} U(i) + 1/2[U(2) + U(14) + U(5,7,8,9) + U(11,12,13) + U(3,4,5,6,10)] \quad (2)$$

$$= 806751$$

$$T2 = 806751/(18 \times 3600) = 12.4 \text{ hrs.}$$

Referring to the actual code and equation (4.3), strategy 3 gives

$$E_{\text{total}} = \sum_{i=1}^{14} U(i) + [1/3U(14) + 1/10U\{(5,7,8,9)\} + 1/25\{U(11,12,13,14)\} + 12/27\{U(3,4,\dots,14)\} + 4/28\{U(2,3,\dots,14)\}] \quad (3)$$

$$= 977778$$

$$T3 = \frac{977778}{18 \times 3600} = 15.08 \text{ hrs.}$$

## 2. Program: CIS 313-L2-TC0671

The structure chart for this program is shown in Section 3.7.

Strategy 1

$$\begin{aligned}
 E(\text{Sub } 10) &= E(\text{Sub}(10,13)) \\
 &= U(10) + U(13) + KU(13) \\
 E(\text{Sub } 9) &= E(\text{Sub}(9,10,12,13)) \\
 &= U(9) + U(12) + U(13) + E(\text{Sub } 10) + K[U(10,12,13)] \\
 E(\text{Sub } 3) &= E(\text{Sub}(3,4,6,7,8)) \\
 &= U(3) + U(6) + U(7) + U(8) + U(4) + K[U(4,6,7,8)] \\
 E(\text{Sub } 2) &= U(2) + E(\text{Sub } 3) + U(4) + U(5) + E(\text{Sub } 9) + U(11) + K[U(3,4,\dots,13)] \\
 E_{\text{total}} &= U(1) + E(\text{Sub } 2) + K[U(2,3,\dots,13)] \\
 &\quad 13 \\
 &= \sum_{i=1} U(i) + K[U(13) + U(10,12,13) + U(4,6,7,8) \\
 &\quad \quad \quad + U(3,4,\dots,13) + U(2,3,\dots,13)] \tag{4}
 \end{aligned}$$

Strategy 2

$$\begin{aligned}
 E(\text{Sub } 10) &= E(\text{Sub}(10,13)) \\
 &= U(10) + U(13) + KU(13) \\
 E(\text{Sub } 9) &= E(\text{Sub}(9,10,12,13)) \\
 &= U(9) + E(\text{Sub } 10) + U(12) + U(13) + K[U(10,12,13)] \\
 E(\text{Sub } 3) &= U(3) + U(4) + U(6) + U(7) + U(8) + K[U(4,6,7,8)] \\
 E(\text{Sub } 2) &= U(2) + U(11) + E(\text{Sub } 3) + U(4) + U(5) + E(\text{Sub } 9) \\
 &\quad \quad \quad + K[U(3,4,5,9,11)] \\
 E_{\text{total}} &= U(1) + E(\text{Sub } 2) + KU(2) \\
 &\quad 13 \\
 &= \sum_{i=1} U(i) + K[U(2) + U(13) + U(4,6,7,8) \\
 &\quad \quad \quad + U(10,12,13) + U(3,4,5,9,11)] \tag{5}
 \end{aligned}$$

The unit efforts for each module are listed below.

Module Number	Effort, E	Union of the Modules	Effort, E
1	39654	U(10,12,13)	182823
2	114327	U(4,6,7,8)	37653
3	19090	U(3,4,...,13)	683082
4	11349	U(2,3,...,13)	989549
5	16414	U(3,4,5,9,11)	215164
6	14203		
7	11769		
8	19257		
9	21307		
10	81104		
11	30401		
12	29491		
13	29491		

Using Equation (4)

$$\begin{aligned}
 E_{\text{total}} &= \sum_{i=1}^{13} U(i) + 1/2[U(13) + U(10,12,13) + U(4,6,7,8) + U(2,3,...,13) \\
 &\quad + U(3,4,...,13)] \\
 &= 437857 + 1/2 \times 1922598 \\
 &= 1399156
 \end{aligned}$$

$$T1 = \frac{1399156}{18 \times 3600} = 21.6 \text{ hrs.}$$

Using Equation (5)

$$E_{\text{total}} = \sum_{i=1}^{13} U(i) + 1/2[U(2) + U(13) + U(10,12,13) + U(4,6,7,8) + U(3,4,5,9,11)]$$

$$= 437857 + 1/2 \times 579458$$

$$= 727586$$

$$T2 = 727586 / (18 \times 3600) = 11.2 \text{ hrs.}$$

Referring to the actual code, Equation (4) along with strategy 3 give

$$E_{\text{total}} = \sum_{i=1}^{13} U(i) + [2/8U(13) + 4/24U(10,12,13) + 2/7U(4,6,7,8) + 7/32U(2,3,\dots,13) + 15/31U(3,4,\dots,13)]$$

(6)

$$= 1033443$$

$$T3 = 1033443 / (18 \times 3600) = 15.9 \text{ hrs.}$$

Program: CIS 313-L2-TC0645

The structure chart for this program was shown in Section 3.7.

Strategy 1

$$\begin{aligned} E(\text{Sub } 9) &= E(\text{Sub}(9,10,12,13)) \\ &= U(9) + U(10) + U(12) + U(13) + K[U(10,12,13)] \end{aligned}$$

$$\begin{aligned} E(\text{Sub } 3) &= E(\text{Sub}(3,6,7,8)) \\ &= U(3) + U(6) + U(7) + U(8) + K[U(6,7,8)] \end{aligned}$$

$$\begin{aligned} E(\text{Sub } 2) &= U(2) + U(11) + E(\text{Sub } 3) + U(4) + U(5) + E(\text{Sub } 9) \\ &\quad + K[U(3,4,\dots,13)] \end{aligned}$$

$$\begin{aligned} E_{\text{total}} &= U(1) + E(\text{Sub } 2) + K[U(2,3,\dots,13)] \\ &= \sum_{i=1}^{13} U(i) + K[U(6,7,8) + U(10,12,13) + U(3,4,\dots,13) \\ &\quad + U(2,3,\dots,13)] \end{aligned} \tag{7}$$

Strategy 2

$$E(\text{Sub } 9) = U(9) + U(10) + U(12) + U(13) + K[U(10,12,13)]$$

$$E(\text{Sub } 3) = U(3) + U(6) + U(7) + U(8) + K[U(6,7,8)]$$

$$\begin{aligned} E(\text{Sub } 2) &= U(2) + U(11) + E(\text{Sub } 3) + U(4) + U(5) \\ &\quad + E(\text{Sub } 9) + K[U(3,4,5,9,11)] \end{aligned}$$

$$\begin{aligned} E_{\text{total}} &= U(1) + E(\text{Sub } 2) + KU(2) \\ &= \sum_{i=1}^{13} U(i) + K[U(2) + U(6,7,8) + U(10,12,13) \\ &\quad + U(3,4,5,9,11)] \end{aligned} \tag{8}$$



The unit efforts for each module are listed below.

Module Number	Effort, E	Union of the Modules	Effort, E
1	31325	U(6,7,8)	71185
2	102441	U(10,12,13)	135153
3	17509	U(3,4,...,13)	653571
4	24823	U(2,3,...,13)	999369
5	49232	U(3,4,5,9,11)	274528
6	24962		
7	25204		
8	42094		
9	29456		
10	56099		
11	26742		
12	26742		
13	26742		

The Equation (7) gives

$$E_{\text{total}} = \sum_{i=1}^{13} U(i) + 1/2[U(6,7,8) + U(10,12,13) + U(3,4,...,13) + U(2,3,...,13)]$$

$$= 483371 + 1/2 \times 1859278 = 1413010$$

$$T1 = 1413010 / (18 \times 3600) = 21.8 \text{ hrs.}$$

Equation (8) gives

$$E_{\text{total}} = \sum_{i=1}^{13} U(i) + 1/2[U(2) + U(6,7,8) + U(10,12,13) + U(3,4,5,9,11)]$$

$$= 483371 + 1/2 \times 583307 = 775025$$

$$T2 = 775025 / (18 \times 3600) = 12 \text{ hrs.}$$

Referring to the actual code, Equation (7) together with strategy 3 give

$$E_{\text{total}} = \sum_{i=1}^{13} U(i) + [(2/6)U(6,7,8) + 4/22U(10,12,13) + 15/28U(3,4,\dots,13) + 7/29U(2,3,\dots,13)] \quad (9)$$

$$= 1121026$$

$$T3 = 1121026/(18 \times 3600) = 17.3 \text{ hrs.}$$

## 4. Program: CIS 313-L3-TC0671

The structure chart for the program is shown in the following diagram.  
The major steps of the calculations are given below.

Strategy 1

$$E(\text{Sub } 33) = U(33) + U(36) + KU(36)$$

$$E(\text{Sub } 32) = U(32) + E(\text{Sub } 33) + U(35) + U(36) + K[U(33,35,36)]$$

$$E(\text{Sub } 26) = U(26) + U(27) + U(29) + U(30) + U(31) + K[U(27,29,30,31)]$$

$$E(\text{Sub } 25) = U(25) + E(\text{Sub } 26) + U(27) + U(28) + E(\text{Sub } 32) + U(34) \\ + K[U(26,27,\dots,36)]$$

$$E(\text{Sub } 2) = U(2) + E(\text{Sub } 25) + K[U(25,26,\dots,36)]$$

$$E(\text{Sub } 23) = U(23) + U(24) + KU(24)$$

$$E(\text{Sub } 8) = U(5) + U(8) + KU(5)$$

$$E(\text{Sub } 13) = U(11) + U(13) + U(15) + K[U(11,15)]$$

$$E(\text{Sub } 22) = U(22) + U(36) + KU(36)$$

$$E(\text{Sub } n) = U(n) + U(14) + E(\text{Sub } 22) + K[U(14,22,36)]$$

$$\text{where } n = 16, 17, \dots, 21$$

$$E(\text{Sub } 12) = U(12) + \sum_{n=16}^{21} E(\text{Sub } n) + K[U(14,16,\dots,22,36)]$$

$$E(\text{Sub } 9) = U(9) + U(11) + E(\text{Sub } 12) + U(4) + K[U(4,11,\text{Sub } 12)] \\ = U(4) + U(9) + U(11) + E(\text{Sub } 12) + K[U(4,11,12,14,16,\dots,22,36)]$$

$$E(\text{Sub } 7) = U(7) + E(\text{Sub } 8) + E(\text{Sub } 9) + U(10) + U(\text{Sub } 13) \\ + U(6) + K[U(4,5,6,8,\dots,22,36)]$$

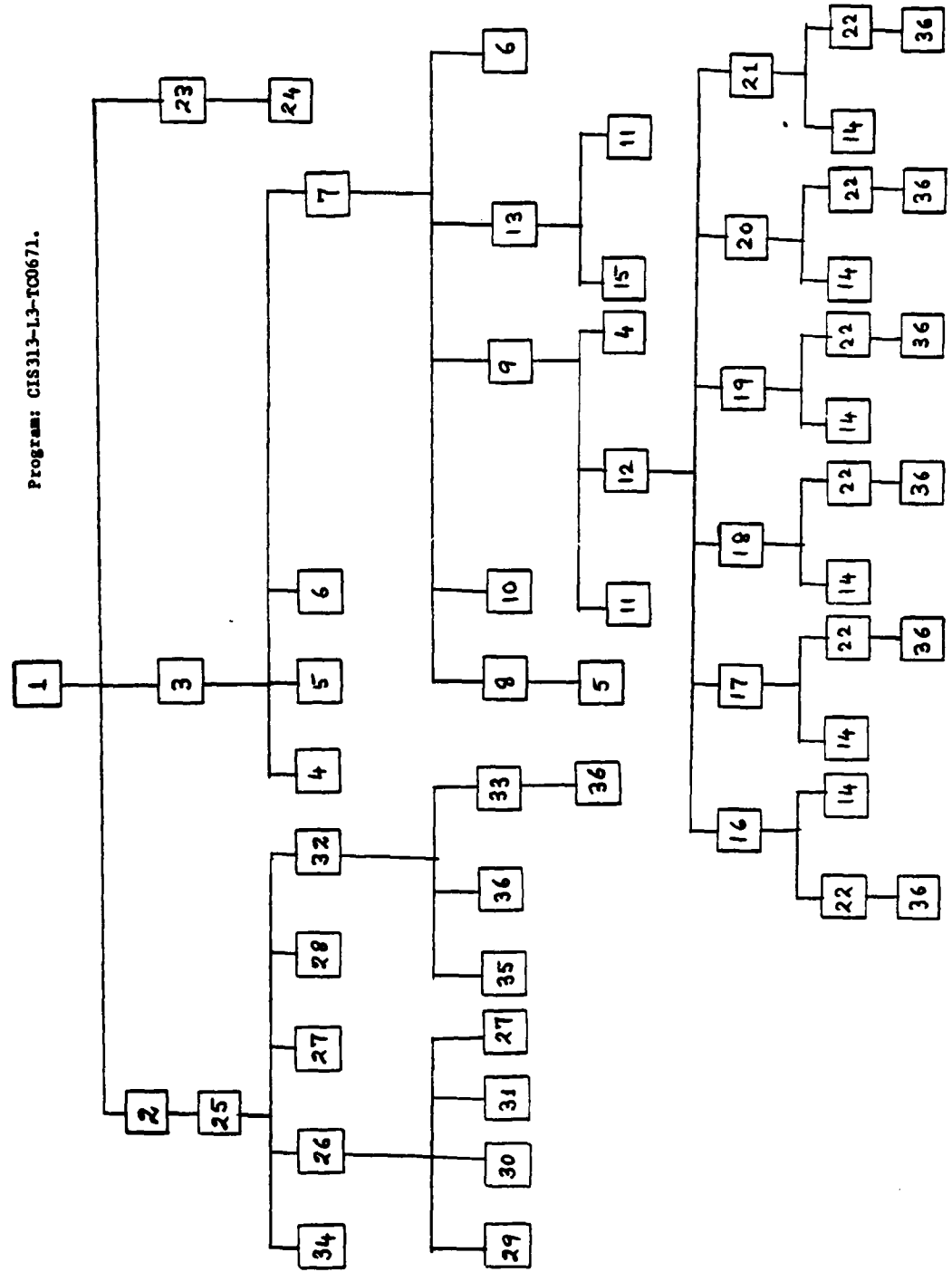
$$E(\text{Sub } 3) = U(3) + U(4) + U(5) + U(6) + E(\text{Sub } 7) + K[U(4,5,\dots,22,36)]$$

$$E_{\text{total}} = U(1) + E(\text{Sub } 2) + E(\text{Sub } 3) + E(\text{Sub } 23) + K[U(2,3,\dots,36)]$$

$$\therefore E_{\text{total}} = \sum_{i=1}^{36} U(i) + K[U(24) + 2U(36) + U(11,15) + U(26,\dots,36) \\ + U(25,\dots,36) + U(33,35,36) + 6U(14,22,36) \\ + U(27,29,30,31) + U(4,11,12,14,16,\dots,22,36) \\ + U(4,5,6,8,\dots,22,36) + U(4,5,\dots,22,36) \\ + U(2,3,\dots,36)]$$

(10)

Program: CIS313-L3-TC0671.



Strategy 2

$$E(\text{Sub } 33) = U(33) + U(36) + KU(36)$$

$$E(\text{Sub } 32) = U(32) + U(35) + U(36) + E(\text{Sub } 33) + K[U(33,35,36)]$$

$$E(\text{Sub } 26) = U(26) + U(27) + U(29) + U(30) + U(31) + K[U(27,29,30,31)]$$

$$E(\text{Sub } 25) = U(25) + U(34) + E(\text{Sub } 26) + U(27) + U(28) + E(\text{Sub } 32) \\ + K[U(26,27,28,32,34)]$$

$$E(\text{Sub } 2) = U(2) + E(\text{Sub } 25) + KU(25)$$

$$E(\text{Sub } 23) = U(23) + U(24) + KU(24)$$

$$E(\text{Sub } 8) = U(5) + U(8) + KU(5)$$

$$E(\text{Sub } 13) = U(13) + U(15) + U(11) + K[U(11,15)]$$

$$E(\text{Sub } 22) = U(22) + U(36) + KU(36)$$

$$E(\text{Sub } n) = U(n) + U(14) + E(\text{Sub } 22) + K[U(14,22)]$$

$$\text{where } n = 16, 17, \dots, 21$$

$$E(\text{Sub } 12) = U(12) + \sum_{n=16}^{21} E(\text{Sub } n) + K[U(16, \dots, 21)]$$

$$E(\text{Sub } 9) = U(9) + U(11) + E(\text{Sub } 12) + U(4) + K[U(4,11,12)]$$

$$E(\text{Sub } 7) = U(7) + E(\text{Sub } 8) + U(10) + E(\text{Sub } 9) + E(\text{Sub } 13) + U(6) \\ + K[U(6,8,9,10,13)]$$

$$E(\text{Sub } 3) = U(3) + U(4) + U(5) + U(6) + E(\text{Sub } 7) + K[U(4,5,6,7)]$$

$$E_{\text{total}} = U(1) + E(\text{Sub } 2) + E(\text{Sub } 3) + E(\text{Sub } 23) + K[U(2,3,23)]$$

$$= \sum_{i=1}^{36} U(i) + K[U(24) + U(25) + U(36) + U(2,3,23) + U(4,11,12) \\ + U(4,5,6,7) + U(33,35,36) + U(5,8) + U(11,15) \\ + U(6,8,9,10,13) + U(27,29,30,31) \\ + U(26,27,28,32,34) + 3\{U(36) + U(14,22)\}] \quad (11)$$

Strategy 3

Referring to the actual code, Equation (10) and the strategy 3 give

$$E_{\text{total}} = \sum_{i=1}^{36} U(i) + [2/8 \cdot 2U(36) + 2/7U(24) + 3/5U(11,15) + 3/37U(25, \dots, 36) + 15/36U(26, \dots, 36) + 4/26U(33, 35, 36) + 2/7U(27, 29, 30, 31) + 2/26U(4, 11, 12, 14, 16, \dots, 22, 36) + 2/34U(4, 5, 6, 8, \dots, 22, 36) + 2/34U(4, \dots, 22, 36) + 9/61U(2, \dots, 36) + 3\{3/16 + 2/16\}U(14, 22, 36)] \quad (12)$$

The unit efforts for each module are listed below.

Module Number	Effort, E	Module Number	Effort, E
1	275416	19	58528
2	15609	20	42012
3	4500	21	40503
4	7104	22	31612
5	5900	23	46382
6	32786	24	23504
7	4750	25	106762
8	25381	26	17687
9	35050	27	10280
10	24527	28	14968
11	49384	29	12942
12	13135	30	10677
13	48179	31	17643
14	63715	32	42536
15	1550	33	81104
16	64625	34	30401
17	30309	35	29491
18	59524	36	30792

Union of the Modules	Effort, E
U(11,15)	50000
U(14,22,36)	210494
U(33,35,36)	182823
U(27,29,30,31)	32981
U(26,...,36)	641582
U(25,...,36)	995663
U(4,11,12,14,16,...,22,36)	653081
U(4,5,6,8,9,...,22,36)	1126370
U(4,...,22,36)	1309907
U(2,3,...,36)	4417377
U(5,8)	39783
U(2,3,23)	171863
U(4,11,12)	139727
U(14,22)	145758
U(4,5,6,7)	171860
U(6,8,9,10,13)	46592
U(26,27,28,32,34)	206592

The use of the numerical values from these tables into Equations (10), (11) and (12) yields the following values of  $E_{\text{total}}$  and Test.

Using Equation (10), with  $K=1/2$

$$E_{\text{total}} = 1409268 + 5389964 \\ = 6799232$$

$$T1 = 6799232 / (18 \times 3600) = 105 \text{ hrs.}$$

Using Equation (11), with  $K=1/2$

$$E_{\text{total}} = 1409268 + 1099731 \\ = 2508999$$

$$T2 = 2508999 / (18 \times 3600) = 38.7 \text{ hrs.}$$

Using Equation (12)

$$\begin{aligned} E_{\text{total}} &= 1409268 + 1486684 \\ &= 2895952 \end{aligned}$$

$$T3 = 2895952 / (18 \times 3600) = 44.7 \text{ hrs.}$$



## 5. Program: CIS 313-L3-TC0645

The structure chart for the program is shown in the following diagram.  
The major steps of the derivation are given below.

Strategy 1

$$E(\text{Sub } 3) = U(3) + U(6) + U(7) + U(8) + K[U(6,7,8)]$$

$$E(\text{Sub } 9) = U(9) + U(10) + U(12) + U(13) + K[U(10,12,13)]$$

$$E(\text{Sub } 2) = U(2) + E(\text{Sub } 3) + U(4) + U(5) + E(\text{Sub } 9) + U(11) \\ + K[U(3,4,\dots,13)]$$

$$E(\text{Sub } 31) = U(31) + U(32) + KU(32)$$

$$E(\text{Sub } 19) = U(15) + U(19) + KU(15)$$

$$E(\text{Sub } 23) = U(23) + U(30) + KU(30)$$

$$E(\text{Sub } n) = U(n) + U(29) + U(30) + K[U(29,30)] \\ \text{where } n = 24, 25, 26, 27, 28$$

$$E(\text{Sub } 22) = U(22) + E(\text{Sub } 23) + \sum_{n=24}^{28} E(\text{Sub } n) + K[U(23,24,\dots,30)]$$

$$E(\text{Sub } 20) = U(20) + U(14) + U(15) + E(\text{Sub } 22) + K[U(14,15,22,\dots,30)]$$

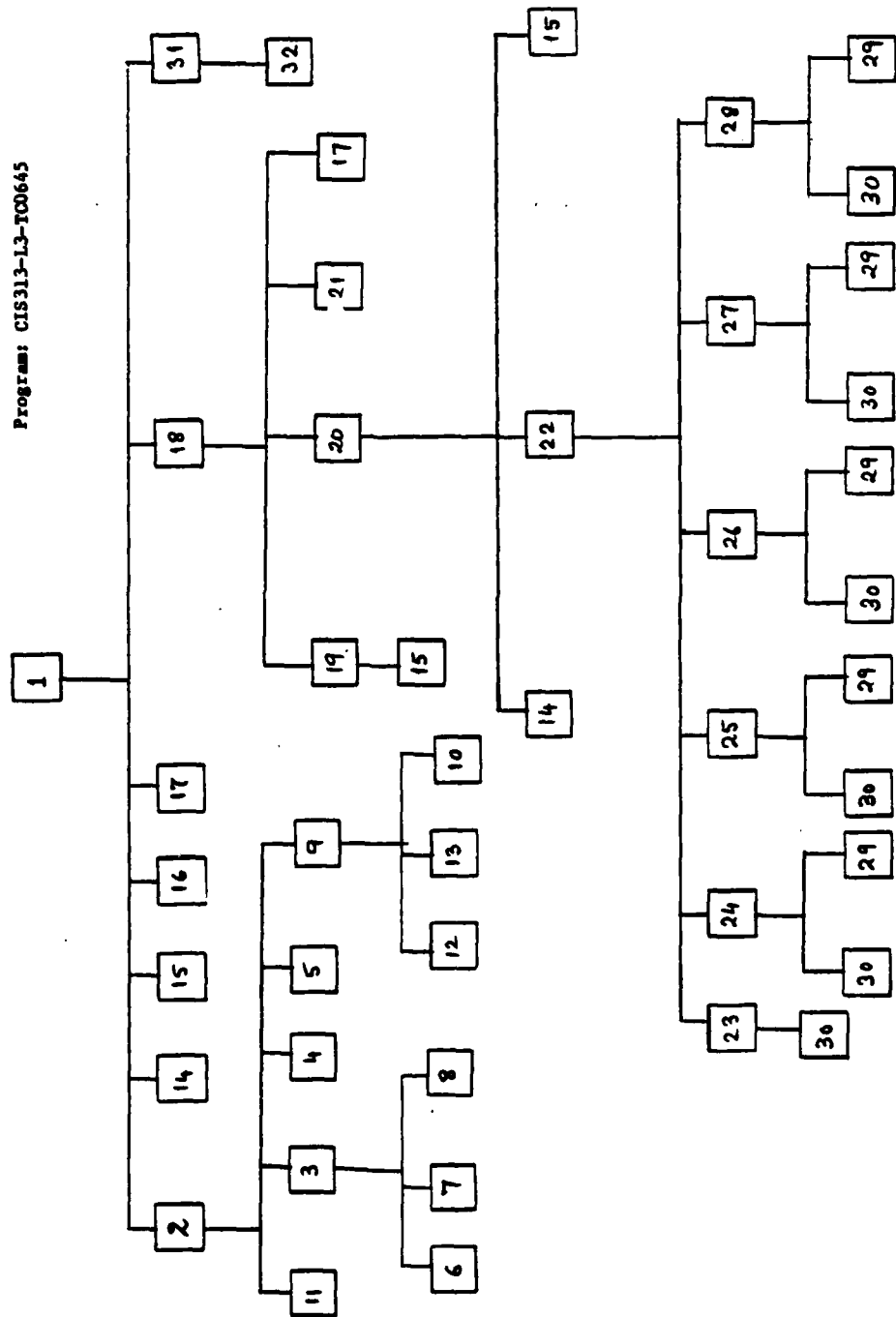
$$E(\text{Sub } 18) = U(18) + E(\text{Sub } 19) + E(\text{Sub } 20) + U(21) + U(17) \\ + K[U(14,15,17,19,\dots,30)]$$

$$E_{\text{total}} = U(1) + E(\text{Sub } 2) + \sum_{i=14}^{17} U(i) + E(\text{Sub } 18) + E(\text{Sub } 31) \\ + K[U(2,3,\dots,32)]$$

$$= \sum_{i=1}^{32} U(i) + K[U(15) + U(30) + U(32) + 5U(29,30) \\ + U(6,7,8) + U(10,12,13) + U(3,4,\dots,13) \\ + U(14,15,22,\dots,30) + U(14,15,17,19,\dots,30) \\ + U(2,3,\dots,30)]$$

(13)

Program: CIS313-L3-TC0645



Strategy 2

$$E(\text{Sub } 3) = U(3) + U(6) + U(7) + U(8) + K[U(6,7,8)]$$

$$E(\text{Sub } 9) = U(9) + U(10) + U(12) + U(13) + K[U(10,12,13)]$$

$$E(\text{Sub } 2) = U(2) + U(11) + E(\text{Sub } 3) + U(4) + U(5) + E(\text{Sub } 9) \\ + K[U(3,4,5,9,11)]$$

$$E(\text{Sub } 31) = U(31) + U(32) + KU(32)$$

$$E(\text{Sub } 19) = U(15) + U(19) + KU(15)$$

$$E(\text{Sub } 23) = U(23) + U(30) + KU(30)$$

$$E(\text{Sub } n) = U(n) + U(29) + U(30) + KU(29,30) \\ \text{where } n = 24, \dots, 28$$

$$E(\text{Sub } 22) = U(22) + E(\text{Sub } 23) + \sum_{n=24}^{28} E(\text{Sub } n) + K[U(23, \dots, 28)]$$

$$E(\text{Sub } 20) = U(20) + U(14) + U(15) + E(\text{Sub } 22) + K[U(14,15,22)]$$

$$E(\text{Sub } 18) = U(18) + E(\text{Sub } 19) + E(\text{Sub } 20) + U(21) + U(17) \\ + K[U(17,19,20,21)]$$

$$E_{\text{total}} = U(1) + E(\text{Sub } 2) + \sum_{i=14}^{17} U(i) + E(\text{Sub } 18) + E(\text{Sub } 31) \\ + K[U(2,14, \dots, 18,31)]$$

$$= \sum_{i=1}^{32} U(i) + K[U(15) + U(32) + 5U(29,30) + U(6,7,8) \\ + U(10,12,13) + U(14,15,22) + U(17,19,20,21) \\ + U(23, \dots, 28) + U(3,4,5,9,11) \\ + U(2,14, \dots, 18,31)]$$

(14)

Strategy 3

Referring to the actual code, Equation (13) and strategy 3 give

$$E_{\text{total}} = \sum_{i=1}^{32} U(i) + [0/3U(15) + 3/3U(30) + 2/6U(32) + 2/7U(6,7,8) + 4/19U(10,12,13) + 15/29U(3,\dots,13) + 7/22U(14,15,22,\dots,30) + 2/23U(14,15,17,19,\dots,30) + 15/50U(2,\dots,32) + \{2 \cdot 4/8 + 3 \cdot 5/8\}U(29,30)] \quad (15)$$

The unit efforts for each module are listed below.

Module Number	Effort, E	Module Number	Effort, E
1	254139	17	30120
2	104156	18	16039
3	17509	19	22697
4	24708	20	112753
5	51208	21	92941
6	25185	22	14133
7	25407	23	137424
8	43104	24	65552
9	78148	25	86413
10	48888	26	85116
11	845916	27	65552
12	26742	28	63435
13	26742	29	42038
14	24574	30	51637
15	5527	31	103208
16	5732	32	16796

Union of the Modules	Effort, E
U(29,30)	84423
U(6,7,8)	74503
U(10,12,13)	116971
U(3,4,...,13)	730000
U(14,15,22,...,30)	643416
U(14,15,17,19,...,30)	1188818
U(2,3,...,32)	5250384
U(14,15,22)	89934
U(17,19,20,21)	265483
U(23,...,28)	283347
U(3,4,5,9,11)	231504
U(2,14,...,18,31)	677593

The use of these numerical values into Equations (13), (14) and (15) yields the following values of  $E_{total}$  and Test.

Using Equation (13), with  $K=1/2$

$$\begin{aligned} E_{total} &= 2613539 + 4250083 \\ &= 6863622 \end{aligned}$$

$$T1 = 6863622 / (18 \times 3600) = 105.9 \text{ hrs.}$$

Using Equation (14), with  $K=1/2$

$$\begin{aligned} E_{total} &= 2613539 + 1091886 \\ &= 3705425 \end{aligned}$$

$$T2 = 3705425 / (18 \times 3600) = 57 \text{ hrs.}$$

Using Equation (15),

$$\begin{aligned} E_{total} &= 2613539 + 2606661 \\ &= 5220200 \end{aligned}$$

$$T3 = 5220200 / (18 \times 3600) = 80 \text{ hrs.}$$

## 6. Program: CIS 313-L3-TC0622

The structure chart for this program is shown in the following diagram. The major steps of the calculations are described below.

Strategy 1

$$E(\text{Sub } 3) = U(2) + U(3) + KU(2)$$

$$E(\text{Sub } 5) = U(5) + U(6) + U(7) + U(8) + U(9) + K[U(6,7,8,9)]$$

$$E(\text{Sub } 11) = U(11) + U(12) + U(13) + U(14) + K[U(12,13,14)]$$

$$E(\text{Sub } 4) = U(4) + E(\text{Sub } 3) + E(\text{Sub } 5) + U(9) + U(10) \\ + E(\text{Sub } 11) + K[U(2,3,5,6,\dots,14)]$$

$$E(\text{Sub } 33) = U(33) + U(34) + KU(34)$$

$$E(\text{Sub } 19) = U(16) + U(19) + KU(16)$$

$$E(\text{Sub } 21) = U(21) + U(22) + U(24) + K[U(22,24)]$$

$$E(\text{Sub } 35) = U(13) + U(35) + KU(13)$$

$$E(\text{Sub } n) = U(n) + U(31) + U(22) + E(\text{Sub } 35) + U(36) \\ + K[U(13,22,31,35,36)]$$

$$\text{where } n = 25, 26, 28, 29, 30$$

$$E(\text{Sub } 27) = U(27) + U(31) + U(32) + E(\text{Sub } 35) + U(36) + U(22) \\ + K[U(13,22,31,32,35,36)]$$

$$E(\text{Sub } 23) = U(23) + \sum_{j=25}^{30} E(\text{Sub } j) + K[U(13,22,25,\dots,31,35,36)]$$

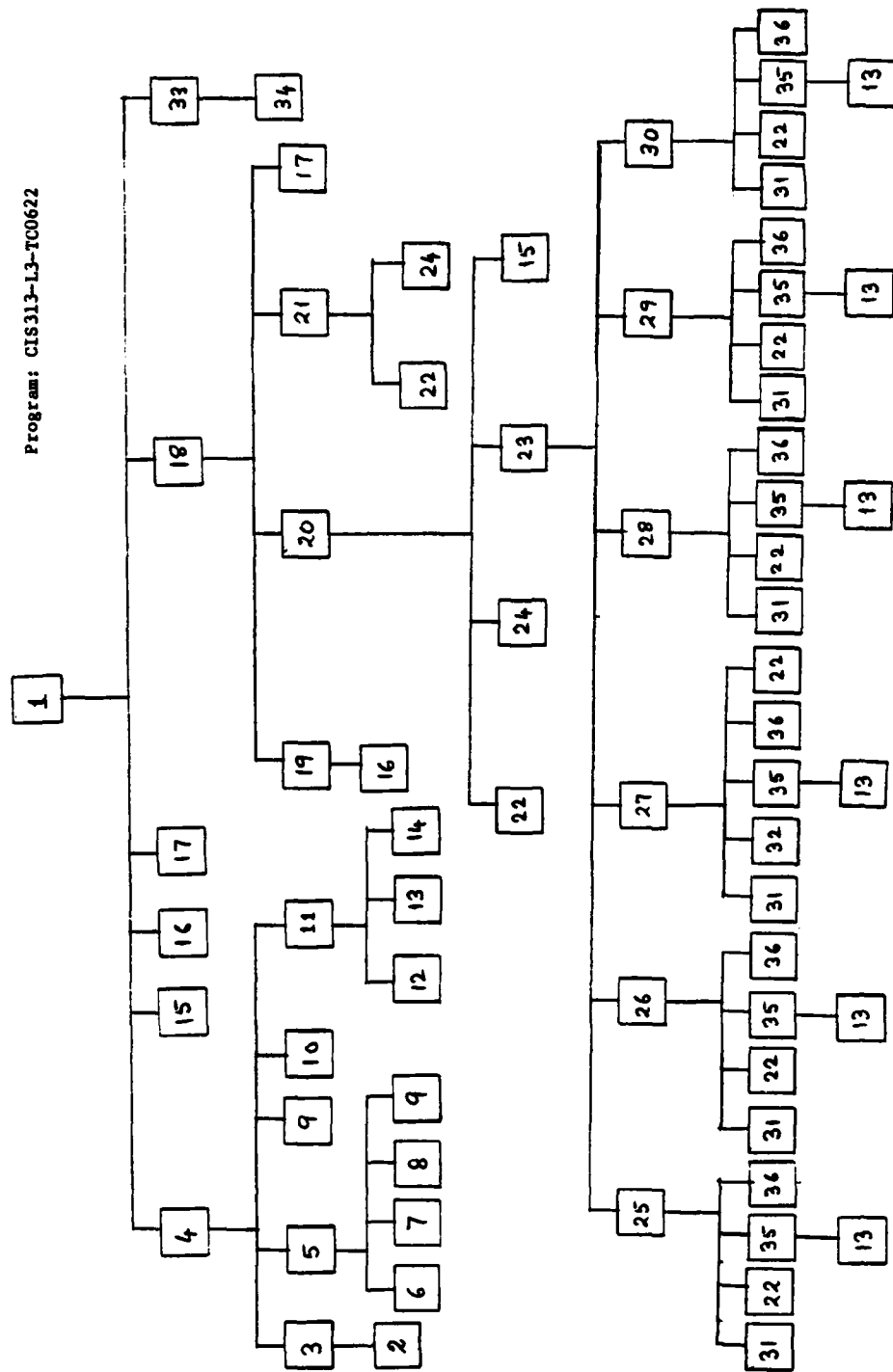
$$E(\text{Sub } 20) = U(20) + U(22) + U(24) + E(\text{Sub } 23) + U(15) \\ + K[U(13,15,22,\dots,31,35,36)]$$

$$E(\text{Sub } 18) = U(18) + E(\text{Sub } 19) + E(\text{Sub } 20) + E(\text{Sub } 21) + U(17) \\ + K[U(13,15,16,17,19,\dots,31,35,36)]$$

$$E_{\text{total}} = U(1) + E(\text{Sub } 4) + U(15) + U(16) + U(17) + E(\text{Sub } 18) \\ + E(\text{Sub } 33) + K[U(2,3,\dots,36)]$$

$$= \sum_{i=1}^{36} U(i) + K[U(2) + U(13) + U(34) + U(12,13,14) \\ + U(6,7,8,9) + U(2,3,5,\dots,14) \\ + U(13,22,31,32,35,36) + 5U(13,22,31,35,36) \\ + U(13,22,25,\dots,31,35,36) + U(13,15,22,\dots,31,35,36) \\ + U(13,15,16,17,19,\dots,31,35,36) \\ + U(2,3,\dots,36)] \quad (16)$$

Program: CIS313-L3-TC0622



Strategy 2

$$E(\text{Sub } 3) = U(2) + U(3) + KU(2)$$

$$E(\text{Sub } 5) = U(5) + U(6) + U(7) + U(8) + U(9) + K[U(6,7,8,9)]$$

$$E(\text{Sub } 11) = U(11) + U(12) + U(13) + U(14) + K[U(12,13,14)]$$

$$E(\text{Sub } 4) = U(4) + E(\text{Sub } 3) + E(\text{Sub } 5) + U(9) + U(10) + E(\text{Sub } 11) \\ + K[U(3,5,9,10,11)]$$

$$E(\text{Sub } 33) = U(33) + U(34) + KU(34)$$

$$E(\text{Sub } 19) = U(16) + U(19) + KU(16)$$

$$E(\text{Sub } 21) = U(21) + U(22) + U(24) + K[U(22,24)]$$

$$E(\text{Sub } 35) = U(13) + U(35) + KU(13)$$

$$E(\text{Sub } n) = U(n) + U(31) + U(22) + E(\text{Sub } 35) + U(36) \\ + K[U(22,31,35,36)]$$

$$\text{where } n = 25, 26, 28, 29, 30$$

$$E(\text{Sub } 27) = U(27) + U(31) + U(32) + E(\text{Sub } 35) + U(36) + U(22) \\ + K[U(22,31,32,35,36)]$$

$$E(\text{Sub } 23) = U(23) + \sum_{j=25}^{30} E(\text{Sub } j) + K[U(25, \dots, 30)]$$

$$E(\text{Sub } 20) = U(20) + U(22) + U(24) + E(\text{Sub } 23) + U(15) \\ + K[U(15,22,23,24)]$$

$$E(\text{Sub } 18) = U(18) + E(\text{Sub } 19) + E(\text{Sub } 20) + E(\text{Sub } 21) \\ + U(17) + K[U(17,19,20,21)]$$

$$E_{\text{total}} = U(1) + E(\text{Sub } 4) + U(15) + U(16) + U(17) + E(\text{Sub } 18) \\ + E(\text{Sub } 33) + K[U(4,15, \dots, 18,33)]$$

$$= \sum_{i=1}^{36} U(i) + K[U(2) + U(13) + U(16) + U(34) \\ + U(22,24) + U(6,7,8,9) + U(12,13,14) \\ + 5U(22,31,35,36) + U(3,5,9,10,11) \\ + U(15,22,23,24) + U(22,31,32,35,36) \\ + U(17,19,20,21) + U(25,26, \dots, 30)]$$

(17)



Strategy 3

Referring to the actual code, Equation (16) and strategy 3 give

$$\begin{aligned}
 E_{\text{total}} = \sum_{i=1}^{36} U(i) &+ [1/2U(2) + 2/8U(13) + 2/7U(34) \\
 &+ 2/6U(6,7,8,9) + 5/21U(12,13,14) \\
 &+ 13/41U(2,3,5,6,\dots,14) \\
 &+ 5/19U(13,22,31,32,35,36) \\
 &+ 1/24U(13,22,25,\dots,32,35,36) \\
 &+ 6/26U(13,15,22,\dots,32,35,36) \\
 &+ 2/31U(13,15,16,17,19,\dots,32,35,36) \\
 &+ 13/52U(2,3,\dots,36) \\
 &+ 5 \times 5/18U(13,22,31,35,36)]
 \end{aligned}
 \tag{18}$$

The unit efforts for each module are listed below.

Module Number	Effort, E	Module Number	Effort, E
1	254810	19	37967
2	3950	20	54702
3	17641	21	83802
4	84206	22	77958
5	20724	23	18400
6	6833	24	48179
7	7184	25	133481
8	10774	26	70769
9	5362	27	44601
10	12442	28	85176
11	106383	29	70769
12	18387	30	68746
13	22949	31	30164
14	55706	32	44229
15	18087	33	55758
16	29017	34	18486
17	48179	35	15638
18	48179	36	4818

Union of the Modules	Effort, E
U(6,7,8,9)	27626
U(12,13,14)	127210
U(13,22,31,35,36)	187760
U(13,22,31,32,35,36)	246134
U(13,22,25,...,31,35,36)	580819
U(13,15,22,...,31,35,36)	1111212
U(13,15,16,17,19,...,31,35,36)	1706111
U(2,3,...,36)	5397419
U(2,3,5,...,14)	712167
U(22,24)	82369
U(22,31,35,36)	154825
U(22,31,32,35,36)	207027
U(17,19,20,21)	223411
U(15,22,23,24)	233153
U(25,...,30)	348113
U(3,5,9,10,11)	308263

The use of these numerical values into Equations (16), (17) and (18) yields the following values of  $E_{total}$  and Test.

Using Equation (16), with  $K=1/2$

$$E_{total} = 1734458 + 5446441 = 7180899$$

$$T1 = 7180899 / (18 \times 3600) = 110 \text{ hrs.}$$

Using Equation (17), with  $K=1/2$

$$E_{total} = 1734458 + 1202849 = 2937307$$

$$T2 = 2937307 / (18 \times 3600) = 45 \text{ hrs.}$$

Using Equation (18)

$$E_{total} = 1734458 + 2343908 = 4078366$$

$$T3 = 4078366 / (18 \times 3600) = 63 \text{ hrs.}$$

END

FILMED

ADTIC